LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Mobiilisovelluksen kehittäminen Ionic-sovelluskehyksen avulla

Case: Duty Officer APP

Pöllänen Ville

2018 Laurea

Laurea-ammattikorkeakoulu

Case: Duty Officer APP

Pöllänen Ville
Tietojenkäsittelyn koulutus
Opinnäytetyö
Huhtikuu, 2018

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutus
Tietojenkäsittelyn koulutus (AMK)

Pöllänen Ville

Mobiilisovelluksen kehitys Ionic-sovelluskehyksen avulla - Case: Duty Officer APP

Vuosi 2018 Sivumäärä 46 + 5

Opinnäytetyön tavoitteena oli toteuttaa IT-alan yritykselle päivystyssovellus, jonka avulla voidaan seurata muun muassa eri ohjelmien virheilmoituksia. Sovelluksen piti olla mahdollisimman responsiivinen ja sitä haluttiin käyttää puhelimella ja eri selaimilla. Sovelluksen on tarkoitus helpottaa tietokantaan tulevien viestien seurantaa. Sovellus tehtiin vain yrityksen sisäiseen käyttöön ja se toteutettiin Ionic-sovelluskehyksen avulla. Sovelluksen kehitys aloitettiin visuaalisesta käyttöliittymämallista ilman mitään aiempaa mobiilikehitys kokemusta. Projektin alussa ei ollut määrittelyä, vaan se tehtiin jälkikäteen käyttäjien kanssa pidettävissä palavereissa.

Opinnäytetyössä menetelminä käytettiin haastattelua ja toiminnallisia menetelmiä. Toiminnallisia menetelmiä käytettiin, koska opinnäytetyö oli projekti yritykselle ja haastattelua taas käytettiin puuttuvan tiedon keräämiseen ja palautekyselyyn. Eniten ongelmia sovelluksen kehityksessä aiheutti VirtualScroll-ominaisuus ja reaaliaikainen tiedonhaku tietokannasta. Projektin aikana VirtualScroll-ongelma ratkaistiin pagination-lisäosalla ja siihen tehdyllä muutoksella. Reaaliaikaisuus päätettiin siirtää jatkokehitykseen. Projektin lopputuloksena syntyi skaalautuva mobiili- ja web-selaimilla toimiva sovellus. Palautekyselystä ilmeni näiden käyttäjien olevan tyytyväisiä sovelluksen nykyiseen tilanteeseen. Sovelluksessa on kuitenkin paljon jatkokehitys mahdollisuuksia, kuten reaaliaikainen tietojenhaku tietokannasta tai muiden lisäominaisuuksien toteutus.

Asiasanat: Ionic, Responsiivinen, Hybridisovellus, Mobiilisovellus

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Pöllänen Ville

Mobile Application Development with Ionic Framework – Case: Duty Officer App

Year 2018 Pages 46 + 5

The purpose of this Bachelor's thesis was to make a duty officer application for an IT industry company, which can be used to monitor error messages from different programs. Application needed to be as responsive as it can be and the company also wanted the application to be usable with phones and different browsers. The purpose of the application is to help the duty officers to check database information. The application was made only for internal usage and it was made with the help of Ionic Framework. Development of the application started with the visual operation system design, without any earlier knowledge or experience of mobile development. In the beginning the project did not have any definition process, rather it was defined afterwards in meetings with the users.

Interviews and operational methods were used in the thesis. Operational methods were used because the thesis was a project and interviews were used to collect missing information as well as in the feedback survey. The biggest problems with the application development were caused by VirtualScroll feature and a real-time data search from database. During the project development the VirtualScroll problem was solved by the pagination add-on and the change made to it. It was decided to move the real-time development further. The result of the project was a responsive application that works on mobile and web browsers. In the feedback survey it was found out that these users were still happy with the application's current situation. However, there are many opportunities for further development such as real-time data retrieval from database or other extra functionality implementations.

Keywords: Ionic, Responsive, Hybrid App, Mobile App

Sisällys

1 Johdanto

Opinnäytetyön aihe oli mobiilisovelluksen kehitys Ionic-sovelluskehyksen avulla ja se tehtiin toimeksiantona IT-alan yritykselle. Yritys halusi toteuttaa päivystyssovelluksen, joka helpottaisi yrityksessä toimivien päivystäjien työntekoa. Päivystyssovelluksen tarkoitus on mahdollistaa tietokannan datajoukkojen näyttäminen sovelluksessa helpommin kuin tietokoneella luettuna. Mobiilikehitys oli minulle uusi asia ja se vaikutti mielenkiintoiselta, joten päätin ottaa projektin vastaan, vaikka minulla ei ollut siitä aiempaa kokemusta. Opinnäytetyön aiheeksi rajautui toteutuksesta kertominen, koska koodista kertovia opinnäytetöitä on paljon. Ajattelin, että prosessista kertominen voisi olla hyvä vaihtoehto varsinkin, jos yritykselle tehtyä koodia ei voida kommentoida opinnäytetyössä, varsinkaan sovelluksen autentikaatio toteutuksesta johtuen. Opinnäytetyön tarkoituksena on kertoa, mitä projektin kehityksen aikana tapahtui ja miten eteen tulleet ongelmat ratkaistiin.

Opinnäytetyön tavoitteena oli toteuttaa mahdollisimman responsiivinen sovellus, joka toimii Android- ja iOS -käyttöjärjestelmillä, sekä verkkoselaimilla. Omina tavoitteinani projektissa oli aikataulu, eli projekti oli tarkoitus saada valmiiksi syyskuussa 2017. Muina tavoitteina sovelluksen kehityksessä oli mahdollisimman helppokäyttöinen käyttöliittymä ja mahdollisimman hyvin optimoitu sovellus puhelimelle ja tietokoneelle. Tarkoitus oli tehdä sovellus, jota oikeasti käytetään ja joka helpottaa tietokannan tietojen seuraamista. Myöhemmin asetin henkilökohtaiseksi tavoitteekseni toteuttaa reaaliaikaisen tiedonhaun tietokannasta sovellukseen.

Opinnäytetyön teoriaosuus koostuu projektiin liittyvistä teknologioista, käsitteistä, lyhenteistä, käytetyistä menetelmistä ja työn lähtökohdista. Opinnäytetyössä on paljon sellaisia termejä ja teknologioita, jotka eivät ole välttämättä kaikille itsestäänselvyyksiä. Ilman teoriaosuutta lukijan on vaikea tietää, mitä opinnäytetyön teknologialla on tehty tai miten ne toimivat. Lyhenteiden ja termien avauksella helpotetaan lukemisen ymmärtämistä. Suurin osa projektin aikana kerätystä tiedoista tulee Ionic-sovelluskehyksen ohjelmointidokumentaatiosta. Ilman kyseistä dokumentaatiota sovelluksen kehitys olisi ollut mahdotonta.

2 Työn lähtökohdat

Tässä kappaleessa kerrotaan projektin käynnistämisestä ja tilaajan eli yrityksen vaatimuksista, jotka sovelluksen tulee täyttää. Kappaleessa kerrotaan myös keskeiset käsitteet, jotka auttavat ymmärtämään projektin ja opinnäytetyön kokonaisuuden.

2.1 Projektin lähtökohta ja tavoitteet

Projektin tavoitteena oli luoda päivystysohjelma IT-alalla toimivan yrityksen sisäiseen käyttöön. Päivystysohjelma hakee tietokannasta päivittäin uusimmat rivit, joista päivystäjät ovat kiinnostuneita ja sen päätarkoituksena on nopeuttaa ohjelmissa tapahtuneisiin virheisiin reagointia. Sovellusta on tarkoitus käyttää pääasiassa päivystyksen yhteydessä helpottamaan päivystäjän normaalia työskentelyä. Haetut tiedot rajoittuvat kyseisen päivän tapahtumiin ja tiettyjen taulujen dataan. Tiedon hakemiseen tietokannasta käytetään C#-koodilla toteutettua rajapintaa.

Sovellusta halutaan käyttää tietokoneilla ja mobiililaitteilla eli sen tulee olla mahdollisimman responsiivinen. Koska päivystyssovellus toteutettiin Ionic-sovelluskehyksellä, se keskittyy enemmän mobiilikehitykseen, mikä voi rajoittaa sovelluksen toimintoja verkkosivuilla. Tämä ei kuitenkaan tarkoita sitä, ettei sovellus toimisi verkkosivuna tai web-sovelluksena.

2.2 Projektin aikataulutus ja vaiheet

Projekti aloitettiin toukokuun loppupuolella ja sen oli tarkoitus valmistua ennen syyskuun loppua 2017. Myös opinnäytetyön tavoitteena oli olla kirjoitettu ennen syyskuun loppua. Yritykselle tehty projekti voidaan jakaa määrittely-, suunnittelu-, kehitys-, testaus- ja tuotantovaiheeseen, joihin oli varattu yhteensä noin neljä kuukautta. Itse sovelluksen kehitys aloitettiin toukokuun loppupuolella. Näihin aikoihin aloitettiin myös opinnäytetyön tekeminen (Kuvio 1.)

Aluksi sovimme työnantajan kanssa, että tekisin projektia 30 % työajasta. Myöhemmin totesimme projektin vievän liian kauan aikaa, jos sitä tehdään vain viikonloppuisin ja yhtenä työpäivänä viikossa. Tämän jälkeen tein sovellusta yritykselle 100 %:sti, kunnes se todettiin julkaisukelpoiseksi.

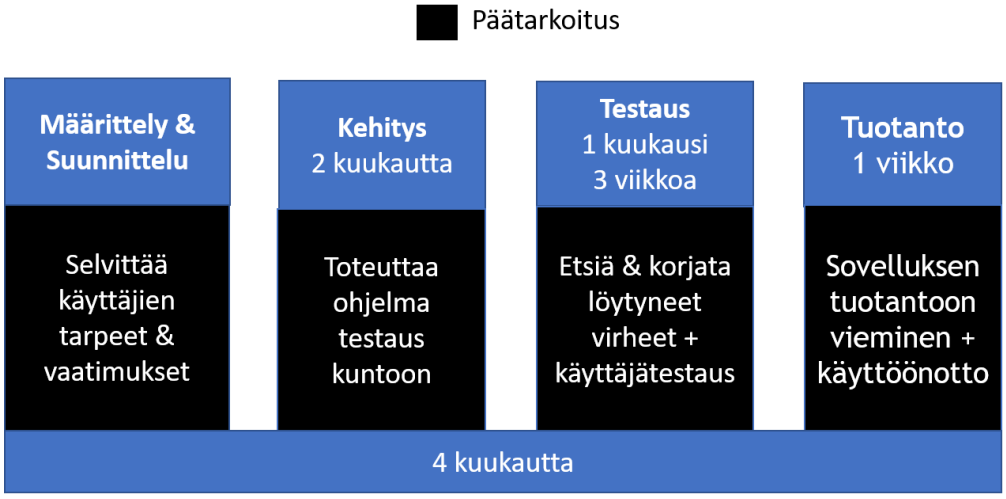
Määrittelyä ei tehty projektille perinteiseen tapaan ensimmäisenä, ennen suunnittelua, vaan se tehtiin sovelluksen käyttäjien kanssa useissa kokouksissa projektin aikana. Sovelluksen määrittelyyn ei siis oltu varattu omaa aikaa vaan sovellus kehittyi käyttäjien toiveiden mukaan. Projektin suunnittelu ja määrittely toteutettiin siis käyttämällä ketterää, prototyypin pohjalta vähitellen kasvavaa projektimallia, jolla pyrittiin eliminoimaan turhan työn tekeminen ja takaamaan mahdollisimman hyvä käyttäjätyytyväisyys (Kuvio 1.)

Suunnittelun aikataulua ei tarkemmin määritelty. Koska projekti aloitettiin korkeatasoisen käyttöliittymä -kuvan perusteella, suunnittelulle ei oltu varattu omaa aikaa (kts. Kuvio 3). Suunnittelua tehtiin kuitenkin vielä sovelluskehityksen aikana. Projektin suunnitelma muuttui kaksi kertaa sovelluksen kehityksen aikana. Ensimmäisen kerran projektin toteutuksen alussa ja toisen kerran projektin loppumetreillä ennen testauksen päättymistä. (Kuvio 1.)

Kehitykseen oli varattu kaksi kuukautta, joiden aikana oli tarkoitus oppia Ionic-sovelluskehyksen käyttö ja luoda toimiva versio ohjelmasta testaukseen. Näinä kahtena kuukautena oli tarkoitus myös kirjoittaa opinnäytetyöosuus kehityksestä valmiiksi. (Kuvio 1.)

Testaukseen oli varattu kuukausi ja kolme viikkoa, johon sisältyi sovelluksen testaus Android-käyttöjärjestelmällä ja eri web-selaimilla. Testauksen aikana myös käyttäjien on tarkoitus testata testiversiota sovelluksesta ja kertoa löydetyistä virheistä, jotka korjataan saman tien. (Kuvio 1.)

Jäljelle jäävä aika oli varattu sovelluksen julkaisemiseen. Sovellus on tuotantokelpoinen, kun se on todettu toimivaksi ja mennyt testeistä läpi. Ionic-projektin tuotantoversiossa ei ole niin paljon eroavaisuuksia projektin verkkosivuille viemisessä testiversioon verrattuna. Testauksessa tulee esille jo kaikki kriittisimmät virheet ja ongelmat, mitkä pitää korjata ennen tuotantoon vientiä. Tuotantoon vientiä ei ole vielä tehty ja tuotantoon vieminen kestäisi enintään viikon. (Kuvio 1.)



Kuvio 1: Suunniteltu sovelluksen toteutuksen aikataulu

2.3 Keskeiset käsitteet ja lyhenteet

Ionic on projektissa käytetty sovelluskehys, jonka kehittämiä komponentteja ja työkaluja käytettiin tämän sovelluksen toteutukseen. Ionic on pääasiassa mobiilikehitykseen tarkoitettu sovelluskehys.

CLI on lyhenne, joka tulee englanninkielisistä sanoista command-line user interface. Komentorivi käyttöliittymä mahdollistaa muun muassa tiedostojen asennuksen, poiston, päivityksen tai muokkauksen.

Sovelluskehys tai ohjelmistokehys voi pohjautua myös toiseen sovelluskehykseen tai hyödyntää sitä esimerkiksi Ionic-sovelluskehys hyödyntää Angular-sovelluskehystä. Sovelluskehys on englanniksi framework. Sovelluskehyksen tarkoitus on nopeuttaa tai helpottaa ohjelmointia, valmiilla komponenteilla tai ominaisuuksilla, joiden toteutus saattaisi kestää puhtaalta pöydältä pitempään.

SQL eli scructured query language, jota käytetään SQL-kyselyiden tekoon. Oikeastaan projektissa käytetään T-SQL eli Transact-SQL:ää, joka on Microsoftin kehittämä SQL-variaatio, joka poikkeaa hieman ominaisuuksiltaan SQL:stä.

VirtualScroll sanaa käytetään kuvaamaan mobiililaitteille tarkoitettua suuren datamäärän yhdellä sivulla näyttämiseen tarkoitettua komponenttia. VirtualScroll lataa näytölle vain tarvittavan määrän dataa, joka vaihtelee näytön koon mukaan. Selatessa alas VirtualScroll päivittää vanhat elementit vastaamaan uutta dataa ilman, että näytetyn datan määrä kasvaa. Eli periaatteessa näytöllä on aina esimerkiksi 5 elementtiä, vaikka tietokannassa olisikin enemmän dataa. Loput datasta näytetään käyttäjälle tarpeen vaatiessa.

InfiniteScroll sanaa käytetään kuvaamaan mobiililaitteille tarkoitettua suuren datamäärän yhdellä sivulla näyttämiseen tarkoitettua komponenttia, joka lataa tietyn määrän dataa per sivu ja lataa sitä lisää aina kun käyttäjä ylittää tietyn datamäärän, kunnes käyttäjä on ladannut kaiken datan tietokannasta, tällä tavalla on mahdollista ladata suuria datamääriä optimaalisesti.

Pagination eli suomeksi sivutus, jolla tarkoitetaan tiettyä verkkosivuille tarkoitettua sivujen jakoa. Yhdelle sivulle ladataan tietyn verran elementtejä ja siitä yli menevät elementit näytetään sivutuksen seuraavalla sivulla. Käytännössä kuitenkin käyttäjän ei tarvitse vaihtaa sivua vaan pagination toimii periaatteessa samalla periaatteella kuin yhden sivun verkkosivu.

CLR Trigger on T-SQL -ominaisuus, jolla voidaan mahdollistaa SQL:n ja ohjelman reaaliaikaisen keskustelun ja ajaa automaattisesti proseduuri tai kysely, esimerkiksi kun tietokantaa päivitetään. (CLR Triggers 2017; CLR Triggers.)

WebSocket on tekniikka tai oikeastaan API, jonka avulla voidaan kuunnella verkkoliikennettä tai lähettää viestejä reaaliaikaisesti. WebSocket mahdollistaa sovelluksen ja palvelimen kaksisuuntaisen yhteyden. Sovelluksen ja palvelimen välinen yhteys katkeaa vasta kun sovelluksen käyttäjä poistuu sivuilta tai ohjelmasta. (ASP.NET SignalR.) Websocket-API:a hyödyntävät muun muassa Socket.IO ja SignalR.

Hybridisovellus nimitystä käytetään yleensä mobiilikehityksessä ja sillä kuvataan sovellusta, joka toimii monella eri käyttöjärjestelmällä, mahdollisesti myös suoraan verkkoselaimella. Hybridisovelluksen etu verrattuna natiivisovelluksen kehitykseen on se, ettei mobiilisovellusta tarvitse tehdä moneen kertaan uudelleen eri käyttöjärjestelmille.

API tulee englannin kielisistä sanoista Application Programming Interface ja sillä tarkoitetaan rajapintaa. Sovelluksessa rajapintaa käytetään hakemaan uusin datajoukko tietokannasta.

HTML on lyhenne ja se muodostuu englannin kielisistä sanoista hypertext markup language. HTML-kieltä käytetään verkkosivujen rakentamiseen ja oikeastaan niiden alustana**.**

CSS on lyhenne englannin kielisistä sanoista cascading style sheets. CSS määrittää verkkosivun ulkonäön.

JavaScript on ohjelmointikieli, jota käytetään yhdessä HTML:n ja CSS:n kanssa rakentamaan verkkosivuja. JavaScript mahdollistaa toiminnallisuuksien lisäyksen HTML-elementteihin.

NPM eli Node.js Package Manager on pakettien hallintaohjelma, jota Node.js hyödyntää. NPM:n avulla voidaan projektiin lisätä sen arkistosta löytyvä paketti. NPM:llä on myös omat verkkosivut, joiden kautta voidaan selata ja ladata erilaisia lisäosia sovelluksiin. NPM on suunniteltu helpottamaan Node.js:n käyttämistä erilaisilla lisäosilla. Periaatteessa toisen kehittäjän jakamaa koodia tai omaa koodia voidaan käyttää hyödyksi erilaisissa projekteissa. Esimerkiksi ngx-pagination lisäosa on ladattu NPM:n kautta. (What is npm? 2017; About Node.js.)

TypeScript perustuu JavaScriptiin ja on omien sanojensa mukaan puhdas JavaScript-kirjasto. TypeScript tukee myös ES6 eli ECMAScript 6 standardia aina ES3 asti eli vanhempaakin standardia voidaan käyttää. TypeScript tiedostot käännetään yleensä puhtaaksi JavaScriptiksi sovelluksen kääntämisen yhteydessä, jonka hoitaa yleensä NPM. (TypeScript.)

SASS eli Syntactically Awesome StyleSheets mahdollistaa esimerkiksi syntaksit, funktiomaisen jäsentelyn ja muuttujat CSS-tyylitysten kanssa. SASS 3 on täysin yhteen sopiva CSS-standardin kanssa. Ionic käyttää SASS:ia CSS-tyylityksiin. SASS 3 -versiosta lähtien SASS käyttää SCSS-tiedostopäätettä. SCSS mahdollistaa täyden tuen CSS 3 -tyyleille, poistamatta SASS:in ominaisuuksia. SCSS-tiedostosta rakennetaan myöhemmin CSS-tiedosto projektin rakennusvaiheen yhteydessä. (Sass (Syntactically Awesome StyleSheets) 2017; Intro to SCSS for Sass Users 2017.)

T-SQL lyhenne Transaction-SQL:stä on Microsoftin tekemä SQL-kieli. T-SQL lisää SQL-toimintoja, jota ei ole mahdollista tehdä normaalissa SQL-kielessä. Vaikka projektissa voitaisiin käyttää muuta SQL-kieltä, on T-SQL Microsoft ympäristössä paras ratkaisu. Syynä tähän on C#-koodin toimivuus paremmin T-SQL -ympäristössä. Myös tietokantapalvelin rajoittaa mitä tekniikkaa tai kieltä voidaan käyttää.

C# on Microsoftin tekemä koodikieli. C# on sukua C ja C++ -kielille. C# toimii hyvin Microsoft ympäristössä esimerkiksi T-SQL:n kanssa. C#-kieli muistuttaa syntaksiltaan paljon Java-kieltä ja se on saanut siitä paljon vaikutteita. C# on osa ASP.NET-sovelluskehystä. Yrityksessä, jossa projektia tehtiin, käytetään paljon C#-koodia, joka määrittää miksi sovelluksen rajapinta on toteutettu ASP.NET-sovelluskehyksellä ja C#-kielellä. (Introduction to the C# Language and the .NET Framework 2015.)

ASP.NET Core on Microsoftin toteuttaman sovelluskehys. ASP.NET Core on suunniteltu yhteensopivaksi juuri Angular-sovelluskehyksen kanssa. Koska Ionic-sovelluskehys perustuu Angular-sovelluskehykseen, voidaan olettaa, että se toimii myös Ionic-sovelluskehyksen kanssa. (Introduction to ASP.NET Core 2017.)

Angular sovelluskehys on Googlen kehittämä. Angular on skaalautuva sovelluskehys. Vanha Angular versio eroaa hieman uudemmasta versiosta, koska se ei käytä TypeScriptiä vaan JavaScriptia. Angular käyttää TypeScriptiä Angular 2-versiosta lähtien. Nykyään uusinta Angularia kutsutaan pelkäksi Angulariksi, jonka nimi oli aiemmin AngularJs. Angularin käyttö eroaa myös Ionic-sovelluskehittimen eri versioissa. Ionic 1 käyttää vanhaa versiota, kun taas uudemmat sovelluskehyksen versiot käyttävät uudempia Angular-versioita.

TFS eli Team Foundation Server mahdollistaa tiimityöskentelyn Microsoft pohjaisessa työympäristössä. TFS:ään laitettua koodia voi toinen ohjelmoija esimerkiksi muokata tai korjata. TFS:ää käytetään ohjelmistokoodin säilömiseen ja hallintaan. (Team Foundation Server.)

TeamCity on JetBrain-yrityksen kehittämä jatkuva integraatio palvelin. Sen avulla voidaan tehdä automaattisia asennuksia yrityksen palvelimelle ja joka mahdollistaa koodin muutosten viennin testipalvelimelle tai tuotantoon. TeamCityn avulla voidaan tarvittaessa myös palata aiempaan versioon sovelluksesta ja se mahdollistaa myös automaattisen komentoketjutuksen. Esimerkiksi NPM-pakettien asentamisen tai komentojen suorittamisen ennen projektin käynnistämistä. (Continuous Integration with TeamCity 2016.)

3 Käytetyt teknologiat

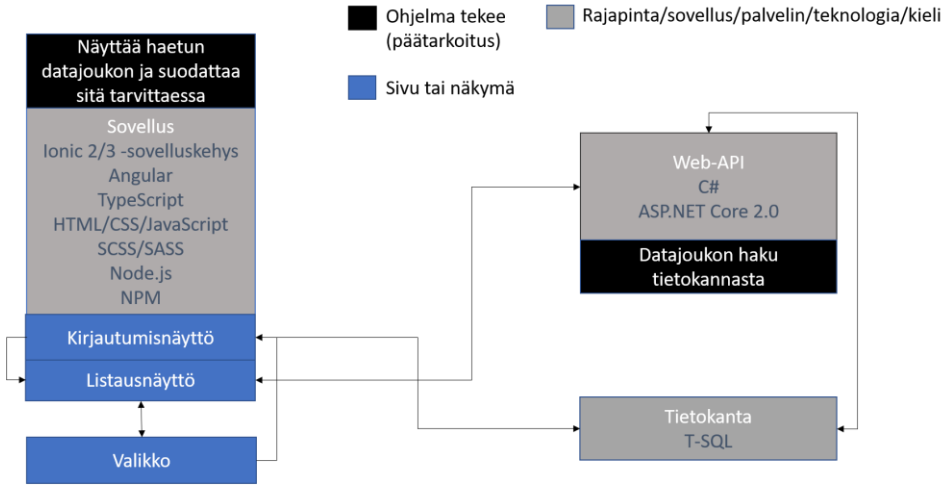
Ionic on mobiilikehitykseen tarkoitettu sovelluskehys, joka perustuu Angular-kirjastoon eli hyödyntää Angularia omassa sovelluskehyksessään. Ionic-sovelluskehyksellä voidaan toteuttaa hybridisovelluksia puhelinjärjestelmille. Projekti luotiin pääosin hyödyntäen Ionicin tarjoamia valmiita komponentteja. Mobiilikehityksessä hybridisovelluksella tarkoitetaan sovellusta, joka mahdollistaa monen käyttöjärjestelmän käytön, ilman mitään muutoksia projektin koodirakenteeseen. Hybridisovelluksen on siis tuettava vähintään useampaa käyttöjärjestelmää, esimerkiksi Android- tai iOS -käyttöjärjestelmiä.

Node.js mahdollistaa JavaScriptin käytön selaimen ulkopuolella, eli palvelimella. Node.js:n avulla voidaan yhdistää front-end ja back-end erilaisten kirjastojen avulla. Node.js seuraa ES6 eli ECMAScript 6 -standardia. Node.js sovelluskehyksellä on mahdollista toteuttaa reaaliaikaisia sovelluksia. Node.js:n palvelinta kehutaan myös nopeaksi ja vakaaksi. (Justyna Rachowicz. When, How And Why Use Node.js Your Backend. 2017.) Ionic-sovelluskehys käyttää NPM-pakettienhallintaa, joka vaatii Node.js:n käytön.

Ionic-sovelluskehys rakennetaan Node.js ja NPM-pakettienhallinnan avulla. NPM oikeastaan rakentaa projektikansion, josta sovellus rakennetaan. NPM-pakettienhallinnan avulla käyttäjän on myös mahdollista päivittää Ionic-sovelluskehys, sen lisäosat ja mahdollista hakea uusia lisäosia sovellukseen. Ionic-sovelluskehys taas on perinyt paljon Angular-sovelluskehyksestä ja sen käyttö mahdollistaa myös Angular-komponenttien käytön Ionic-projekteissa. Ionic sivut taas koostuvat eri komponenteista ja sen rakenteessa käytetään TypeScript-, SASS/SCSS- ja HTML5-teknologioita. NPM rakentaa näiden teknologioiden avulla projektikansioon HTML-, CSS-, JavaScript-koodilla ohjelman, jota voidaan käyttää suoraan sellaisenaan verkkopalvelimilla. Sovelluksen mobiiliversio tosin rakennetaan Cordovan avulla. (Kuvio 2.)

Päivystyssovellus koostuu periaatteessa kolmesta eri sivusta. Näitä sivuja ovat kirjautumisnäyttö, listausnäyttö ja valikko. Kirjautumisnäytöltä pystyy menemään autentikoinnin jälkeen vain listausnäytölle. Valikosta taas pääsee takaisin listausnäytölle ja kirjautumisnäytölle. Listausnäytöllä käyttäjä pystyy kuitenkin käyttämään valikkoa, toisin kuin kirjautumisnäytöltä. (Kuvio 2.)

Web-API eli sovelluksen rajapinta taas hakee käyttäjälle datajoukkoja tietokannasta ja se on toteutettu C#-koodilla. Web-API perustuu uuteen ASP.NET Core 2.0-teknologiaan, joka julkaistiin sovelluksen kehityksen loppuvaiheessa. Periaatteessa rajapinta hoitaa tiedonhaun sovellukseen ja sitä käytetään vain sovelluksen listausnäytöllä, josta käyttäjä pystyy halutessaan hakemaan uudempia datajoukkoja. (Kuvio 2.)



Kuvio 2: Sovelluksen arkkitehtuuri ja käytetyt teknologiat

4 Menetelmät

Opinnäytetyö toteutettiin kehittämistyönä ja siinä käytettiin toiminnallisia menetelmiä ja haastattelua. Toiminnallisia menetelmiä käytettiin, koska sovellus oli yrityksen antama projekti ja tämä opinnäytetyö kertoo sen tekemisestä ja tavoitteista. Haastattelua käytettiin, koska projektin suunnittelu oli jo aloitettu ennen opinnäytetyön aloittamista. Haastattelulla haluttiin myös saada käyttäjien mielipiteitä projektin toteutukseen ja jatkokehitykseen liittyen.

4.1 Toiminnallinen

Toiminnallisen opinnäytetyön ohessa syntyy aina myös tuotos. Tuotoksesta pyritään saamaan mahdollisimman tarkka kuva projektista kertovien kuvien ja tekstin avulla. Koska opinnäytetyön yhteydessä yritykselle tehdään tuotos eli sovellus, voidaan todeta, että opinnäytetyön tekoon kannattaa menetelminä käyttää toiminnallisia menetelmiä. (Airaksinen & Vilkka 2003, 51.)

Kyseinen opinnäytetyö on tehty projektityönä. Projekti on tietyn ajan kestävä prosessi ja se yleensä määritellään ja toteutetaan tietyn aikataulun mukaisesti. Projektin tarkoitus on tietyn tavoitteen toteuttamisen loppuun vienti. Opinnäytetyönä tehty projekti toteutettiin IT-alan yritykselle ja sen aikataulu ja projektin rajaukset olivat sovelluksen käyttäjien ja yrityksen määrittelemiä. Projekti rajattiin koskemaan yrityksen päivystäjiä. (Airaksinen & Vilkka 2003, 47 - 49.)

Toiminnallisen opinnäytetyön toteutuksessa pitää kuitenkin muistaa, että toiminnallisessa opinnäytetyössä on myös tutkimusviestinnän piirteitä. Näitä ominaisuuspiirteitä ovat argumentointi, käsitteiden tai termien määrittely ja käyttö, lähteiden käyttö ja lähdeviittausten merkintä, persoona- ja aikamuotojen tarkoituksen mukainen valinta, tiedon varmuuden asteen ilmaisu ja metateksti. (Airaksinen & Vilkka 2003, 101.)

Argumentoinnilla pyritään vakuuttamaan lukija tekstin oikeellisuudesta, perustelemalla väitteet. Jos varmaa tietoa ei ole, tulee se myös opinnäytetyössä ilmaista. Argumentoinnin tueksi voidaan käyttää muun muassa lähteitä, omia havaintoja tai toisen opinnäytetyön tuloksia. Argumentoinnin tarkoitus on todistaa ovatko opinnäytetyössä väitetyt väitteet totta. (Airaksinen & Vilkka 2003, 102.)

Käsitteet ja termit on määriteltävä, jotta lukijalle on selkeää mitä niillä tarkoitetaan. Tässä opinnäytetyössä käytetään käsitteitä ja termejä, joiden käyttötarkoitus voi muuttua tai sen määrittely voi vaihdella tiettyjen lähteiden mukaan. Myös käytetyt lyhenteet pitää avata lukijalle, mitä niillä tarkoitetaan. Käsitteet ja termit voivat ainakin IT-alalla muuttua ajan mukaan, joten on mahdotonta todentaa ilman niiden määrittelyä mitä niillä tarkoitetaan. Myös näiden lähde on merkittävä, jos niiden määrittely on otettu lähteestä. (Airaksinen & Vilkka 2003, 104.)

Lähteiden käyttö on toiminnallisessa opinnäytetyössä välttämätöntä. Syy tähän on toiminnallisen opinnäytetyön vaatima pohjamateriaali. On vaikeaa todentaa väitteet ilman asiaan kuuluvaa aineistoa tai ilman luotetuista lähteistä saatua tietoa. Lähdemateriaalin käytössä pitää myös varmistaa voidaanko lähdettä käyttää opinnäytetyössä. (Airaksinen & Vilkka 2003, 106 – 107.)

Opinnäytetyön persoonamuodon valintaan vaikuttaa se, mikä on järkevintä tietyssä vaiheessa toiminnallisen opinnäytetyön kerrontaa ajatellen. Yleensä tutkimustyössä kirjan mukaan opinnäytetyö kirjoitetaan neutraalisti ottamatta kantaa tekstiin tai kirjoittajan persoonaan. Yleensä kirjoittaja oman mielipiteen aiheesta antaa vasta tutkielman lopussa. Toiminnallisessa opinnäytetyössä voi käyttää muitakin persoonamuotoja, kunhan ne ovat perusteltuja. Toiminnallisessa työssä ei siis ole määrättyä persoonamuotoa. Aikamuodon valintaan taas vaikuttaa pitkälti mitä tai mistä opinnäytetyössä kerrotaan. Imperfekti sopii esimerkiksi tekemästään työstä kertomiseen. (Airaksinen & Vilkka 2003, 111 - 112, 117.)

Opinnäytetyössä on kerrottava, kuinka varmaa tieto on. Väitteet tulee pystyä todentamaan. Kirjan mukaan varmuuden aste on yksi toiminnallisen opinnäytetyön mittari, jolla todennetaan, kuinka totta opinnäytetyön väitteet ovat. Jos tieto ei ole täysin varmaa, pitää se myös kertoa opinnäytetyössä. (Airaksinen & Vilkka 2003, 123.)

Metatekstillä tarkoitetaan tekstiä tekstissä. Metatekstin tarkoitus on auttaa lukijaa ymmärtämään tekstiä paremmin. Metatekstillä voidaan opinnäytetyössä käyttää viittausta toiseen tekstiin esimerkiksi aiemmassa tai tulevassa kappaleessa. (Airaksinen & Vilkka 2003, 127.)

Toiminnallinen opinnäytetyö kirjoitetaan kuin se olisi kertomus ja se kerrotaan siinä järjestyksessä kuin missä se on tapahtunut. Kertomusta ohjaa muun muassa työn eteneminen ja siinä tehdyt ratkaisut. Koska opinnäytetyön tarkoitus on oppia tekemisestään, toiminnallisessa opinnäytetyössä on myös hyvä kertoa virheistä, onnistumisista ja projektin muutoksista sen kehittymisen aikana. (Airaksinen & Vilkka 2003, 82.)

4.2 Laadullinen haastattelu

Haastattelu on tehokas tutkimusmenetelmä, jolla voidaan kerätä tietoa opinnäytetyöhön. Tässä opinnäytetyössä käytettiin strukturoitua haastattelua suunnittelijan ja käyttäjien haastatteluun. Strukturoitu haastattelu sopii muun muassa asiantuntijoiden haastatteluun. Haastattelun tarkoituksena oli saada tietoa siitä, milloin projekti oli suunniteltu ja projektin määrittely oli aloitettu ja miten se oli toteutettu. Palautekyselyssä käyttäjiltä taas haluttiin palautetta siitä, miten projekti heidän mielestään eteni.

Tutkimushaastatteluilla voidaan selvittää asioita, jotka ovat jo tapahtuneet. Opinnäytetyössä tutkitaan mistä suunnittelu lähti ja mitkä asiat vaikuttivat sovelluksen suunnitteluun. Suunnittelijan haastattelu on pyritty toteuttamaan niin, että haastattelun avulla saadaan vastaus tiettyihin ennalta määritettyihin kysymyksiin. (Hyvärinen & Nikander 2017, 12.)

Usein laadullisissa haastatteluissa mietitään, kuinka monta haastattelua tarvitaan saadakseen maksimaalinen hyöty haastattelusta. Tässä opinnäytetyössä haastattelujen määrän rajoittaa suunnittelijoiden määrä, joita oli käytetty visuaalisen kuvan toteutuksessa ja sovelluksen suunnittelun alkuvaiheessa. Mahdollisia tutkimuskohteita oli siis vain yksi, jos haastattelu halutaan rajata tiettyyn kohderyhmään. Loppuhaastattelussa taas määrää rajoittivat sovelluksen käyttäjät. (Hyvärinen & Nikander 2017, 34.)

Laadullisessa haastattelussa haastattelutilannetta lähestytään yksilöllisenä haastattelutilanteena. Laadullinen haastattelu poikkeaa määrällisestä haastattelusta siten, ettei siinä olla niin kiinnostuneita haastattelujen määrästä vaan jokaiseen haastatteluun kiinnitetään enemmän huomiota. Laadullisessa haastattelussa on myös kiinnostuttu haastatteluun liittyvistä yksityiskohdista. Esimerkiksi haastateltavan kokemuksista, kertomuksista tai näkemyksistä. (Hyvärinen & Nikander 2017, 89 - 90.)

5 Toteutus

Sovelluksen toteutus voidaan jakaa viiteen vaiheeseen: Määrittely-, suunnittelu-, kehitys-, testaus- ja tuotantovaiheeseen (Nuutila, Sinkkonen & Törmä 2009, 31, 42). Projektin lopputulos yleensä muodostuu näiden vaiheiden yhtälöstä. Jos jokin näistä vaiheista on toteutettu huonosti, aiheuttaa se yleensä viivästyksiä sovelluksen julkaisuun tai valmiin version tuottamiseen.

5.1 Suunnittelu

Suunnittelu on yksi osa sovelluksen toteutusta. Suunnittelu tehdään yleensä määrittelyn jälkeen. Sovelluksesta oli jo ehditty toteuttaa visuaalisia suunnitelmia, joten puuttuva tieto haluttiin selvittää haastattelulla. Suunnitelmaan tehtiin myös projektin aikana muutoksia, joista kerrotaan tässä kappaleessa.

5.1.1 Määrittelyn ja suunnittelun lähtökohdat

Aiemmissa yritykselle tehdyissä projekteissa olen saanut yleensä myös tarkemmin määritellyn suunnitteludokumentin ennen sovelluksen kehittämisen aloittamista. Halusin selvittää suunnittelijan kanssa käydyn haastattelun avulla, oliko sovellukseen tehty aiempaa määrittelyä tai miten sovelluksen määritys oli tarkoitus tehdä. Suunnittelijan haastattelu tehtiin strukturoitujen kysymysten avulla (kts. Liite 1).

Määrittelyvaihetta sovelluksesta ei tehty perinteisellä menetelmällä, vaan se toteutettiin projektin kehityksen aikana. Ideana oli saada päivystäjän työtä helpottava sovellus. Projektin aloituksessa käytiin kokous päivystäjien kanssa. Kokouksessa selvitettiin miltä sovelluksen tulisi näyttää ja siitä tehtiin suunniteltu korkeantason käyttöliittymämalli (Kuvio 3). Normaalisti yrityksessä tehdään määrittely ja suunnittelu paljon tarkemmin, ennen kehityksen aloittamista varsinkin isommissa projekteissa. (Suunnittelija 2017.)

Sovelluksen aikana määrittely- ja suunnitteluvaihe tehtiin käyttäjien kanssa käytävien kokouksien avulla. Palavereita pidettiin noin kahden viikon välein ja niihin osallistui sovelluksen käyttäjien lisäksi suunnittelija ja esimiehiä. Määrittelyn puuttuminen tuotti vaikeuksia ulkonäön suunnittelussa. Päivystäjiltä ei oltu kysytty toiveita ennen suunnittelua vaan sovelluksen käyttöliittymästä oli tehty alustava malli miltä sen tulisi näyttää. (Suunnittelija 2017.)

Sovellus ajateltiin aluksi mobiilisovellukseksi. Mobiilisovellus eroaa käytettävältä tilaltaan paljon verkkosivusta. Sovelluksen tavoite kuitenkin oli olla mahdollisimman responsiivinen. Ongelmana oli, että tietokantataulujen sarakkeinen näyttäminen puhelimella oli hieman erilaista kuin taulukon näyttäminen verkkosivuilla. Tämä vaikutti suunnittelijan mukaan paljon mobiiliversion suunnitteluun. (Suunnittelija 2017.)

Visuaalinen suunnitelma tehtiin mahdollisesti 2 vuotta sitten. Tarkkaa aikaa ei kuitenkaan tiedetty tai voitu varmentaa. Suunnittelija ei osannut sanoa vastasiko suunnitelma mobiilikehityksen nykyisiä standardeja. Suunnittelijan mukaan alkuperäistä suunnitelmaa olisi muutettu, jos se olisi aloitettu 2 vuotta myöhemmin. (Suunnittelija 2017.)

Sovelluksen värimaailma sai vaikutteita yrityksen värimaailmasta, yrityksen väriteemoista ja muissa sovelluksissa käytetyistä väreistä. Yrityksen värimaailma on nykyään hieman erilainen, kuin se suunnitelmaa tehtäessä 1-2 vuotta sitten oli ollut. (Suunnittelija 2017.)

Koska kävimme kokouksia käyttäjien, suunnittelijan ja muiden projektiin liittyvien henkilöiden kanssa, ei sovellukseen tulleista muutoksista koitunut ongelmaa. Suunnittelijan mukaan oli hyvä saada sovellus käyttäjien mieleiseksi. Tietenkin sovellukseen tehdyt muutokset esiteltiin käyttäjille ja verrattiin vanhaa uudistettuun versioon. Kysyin myös hakupalkin lisäämisestä ja suunnittelijan mielestä tämä oli myös hyvä lisäys. (Suunnittelija 2017.)

Suunnittelijan mukaan aina olisi parempi aloittaa toteutus heti sen jälkeen, kun projekti on suunniteltu. Varsinkin pitkät aikavälit voivat haitata suunnittelun jatkamista. Asiakassovelluksissa yleensä suunnittelun ja toteutuksen aikavälit eivät ole näin suuria, kuin yrityksen sisäisissä projekteissa saattaa olla. Asiakasprojektit ovat ymmärrettävästi tärkeämpiä. (Suunnittelija 2017.)

Se kuinka paljon haittaa suunnitelman muutokset aiheuttavat, riippuu projektin suuruudesta tai sen laajuudesta. Varsinkin jos suunnittelun ja toteutuksen aikaväli on suuri, voi tämä haitata suunnittelun jatkamista. Asiakkaaltakin voi joskus tulla muutoksia tai toiveita projektin aikana. Projektin koko voi myös vaikuttaa suunnitteluun tehtäviin muutoksiin. (Suunnittelija 2017.)

Suunnitelman muuttuminen riippuu siitä, kuinka pitkällä sovelluksen toteutusvaihe on. Onko se tuotantovalmis vai ei. Määrittely ei kuitenkaan aina voi olla täydellinen. Suunnittelun muuttaminen ei kuitenkaan yleensä ole huono asia. Välillä tulee pieniä ja välillä suuria muutoksia suunnitelmiin. (Suunnittelija 2017.)

Pidin alkuperäisessä suunnitelmassa sovelluksen värimaailmasta, värit eivät hyppineet silmille, eikä sovellusta oltu koristeltu liian useilla eri väreillä. Sovelluksen määrittely on mielestäni mahdollista tehdä myös sovelluksen kehityksen aikana. Olisi tietenkin ollut parempi toteuttaa sovellus heti mallikuvan perusteella, kun se valmistui, kuitenkaan tämä ei ollut mahdollista. Mielestäni haastattelun kysymyksillä saatiin kokonaiskuva puuttuneeseen tietoon. Ilman suunnittelijan haastattelua näitä ei olisi voitu tietää.

5.1.2 Visuaaliset muutokset alkuperäiseen suunnitelmaan

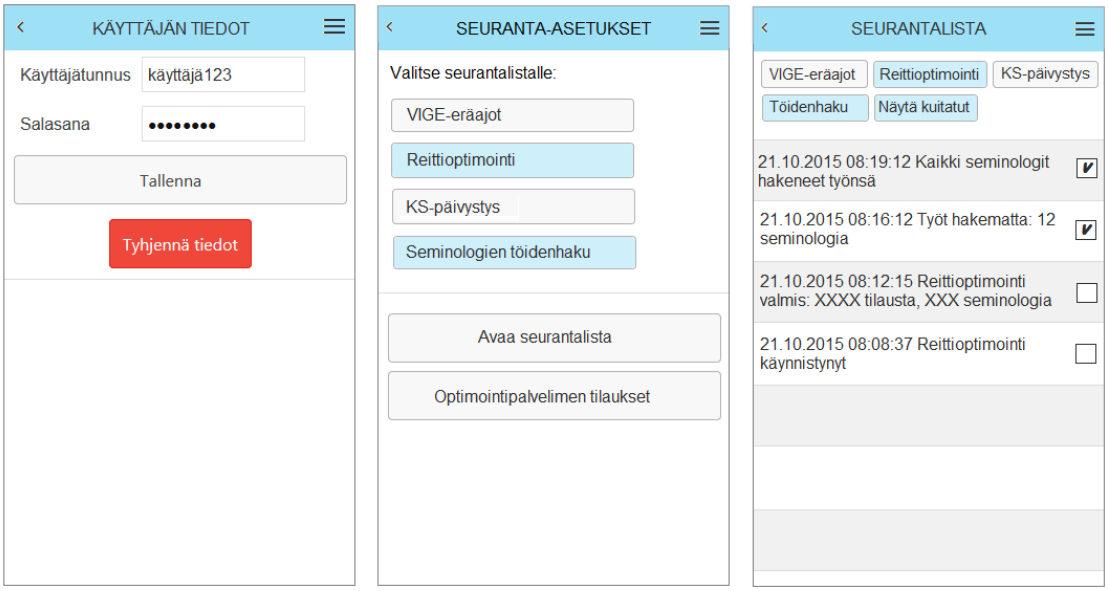
Päivystyssovellus on suunniteltu käytettäväksi mobiililaitteilla. Tämä ei kuitenkaan tarkoita sitä, ettei sovellusta voisi käyttää tietokoneilla. Hyvä suunnittelu helpottaa ohjelman toteuttamista, testausta ja julkaisua.

Käytin suunnittelun oppaana muutamaa kysymystä, jotka auttoivat alkuperäisen suunnitelman parantamisessa. Kysymykset olivat omia kysymyksiäni, joita kysyin itseltäni sovelluksen kehityksen aikana helpottamaan suunnitteluprosessia. Projektin aikana näihin kysymyksiin oli tarkoitus saada vastaus.

- Mitkä ovat käyttäjän tarpeet?
- Mitä sovelluksella halutaan tehdä?
- Mikä ongelma ohjelmalla ratkaistaan?

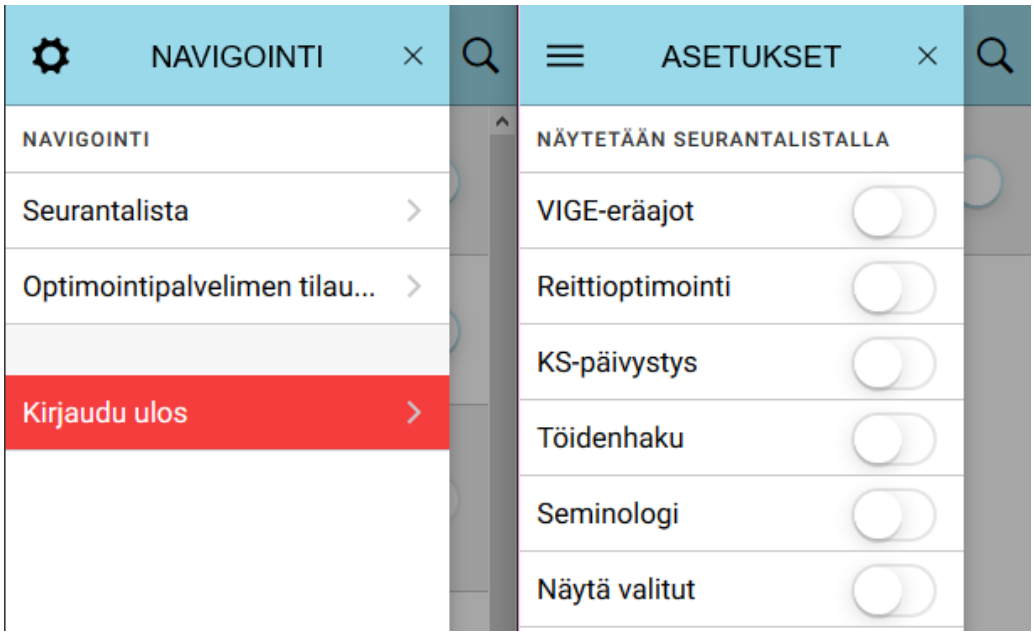
(Nuutila ym. 2009, 20-22.)

Alkuperäistä suunnitelmaa muutettiin hieman projektin aikana (Kuvio 3). Suunnitelmat voivat projektin tekemisen aikana muuttua, eikä valmis ohjelma vastaa aina täysin alkuperäistä suunnitelmaa. Tästä syystä on hyvä tietää mitä ohjelmalla halutaan tehdä ja miten se on järkevintä toteuttaa. Jos suunnitelma on vanha, on järkevintä päivittää vanhaa suunnitelmaa, kuten tässä projektissa tehtiin. Alkuperäisessä suunnitelmassa oli suunniteltu kirjaussivu vasemmalla, keskellä asetukset-sivu ja oikealla seurantalista-sivu. Ymmärtääkseni tarkoituksena kai oli, että seuranta-asetukset -sivu olisi niin sanottu navigointisivu ohjelmalle, johon käyttäjä pääsee kirjautumisen jälkeen (Kuvio 3).



Kuvio 3: Päivityssovelluksen alkuperäinen visuaalinen käyttöliittymämalli

Yksi muutoksista oli navigointisivun poistaminen. Ideana oli toteuttaa sivuvalikkoon navigointi ja asetukset yhden valikon alle. Valikossa oli navigointi- ja asetukset-nappi, joka vaihtaa kuvaa riippuen kummassa valikkoikkunassa käyttäjä oli. Asetuksissa oli myös enemmän valintoja, koska käyttäjät kokivat ne vielä tässä vaiheessa tarpeelliseksi. Kuviosta 3 voi huomata miten kuvioon 4 verrattuna navigointi- ja asetukset-sivu on muuttunut yhteen valikkoon (Kuviot 3 ja 4). Tämä mahdollisti yhden sivun poistamisen sovelluksesta. Kuviossa 4 näkyvä valikko saadaan näkyviin painamalla valikkonappia, joka sijaitsee sovelluksen oikeassa yläkulmassa (Kuvio 4). Itse valikosta pystyi vaihtelemaan navigoinnin ja asetusten välillä painamalla oikealla yläkulmassa olevaa nappia, kun valikko oli aktiivinen. Valikossa oli myös ominaisuus, joka mahdollisti sivukohtaiset asetukset riippuen millä sivulla käyttäjä oli (Kuvio 4). Kuvion 4 oikealla puolella oli navigointivalikko ja vasemmalla taas sivukohtaiset asetukset (Kuvio 4).

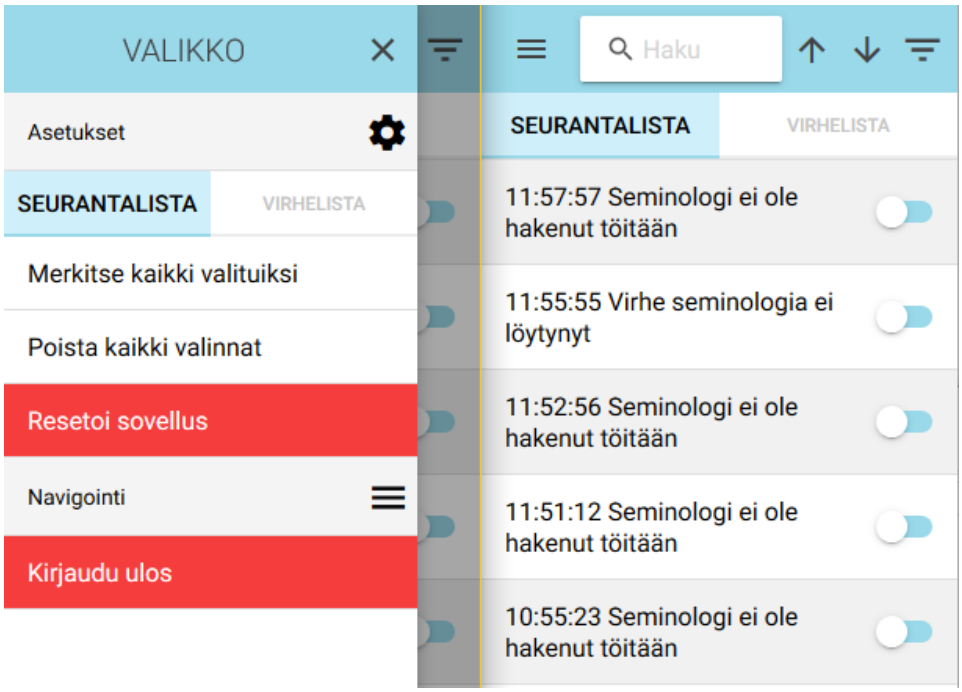


Kuvio 4: Yhdistetty navigointi ja asetukset

Uudistettua navigointia muutettiin vielä projektin lopuksi. Syy tähän oli se, ettei suodatusnappeja tarvittu niin paljon kuin aiemmin oli kuviteltu. Muutoksesta keskusteltiin palaverissa käyttäjien kanssa. Navigointi muutettiin kuitenkin käyttäjäystävälliseksi niin, että käyttäjän ei tarvitse erikseen vaihtaa valikkonäkymää vaan valikkoon listataan otsikoittain asetukset ja navigointi erottimet. Navigointia ei tarvittu enää valikossa, koska se korvattiin paremmalla ominaisuudella ja navigointi siirrettiin listaus-sivu näkymään (kts. Kuvio 5).

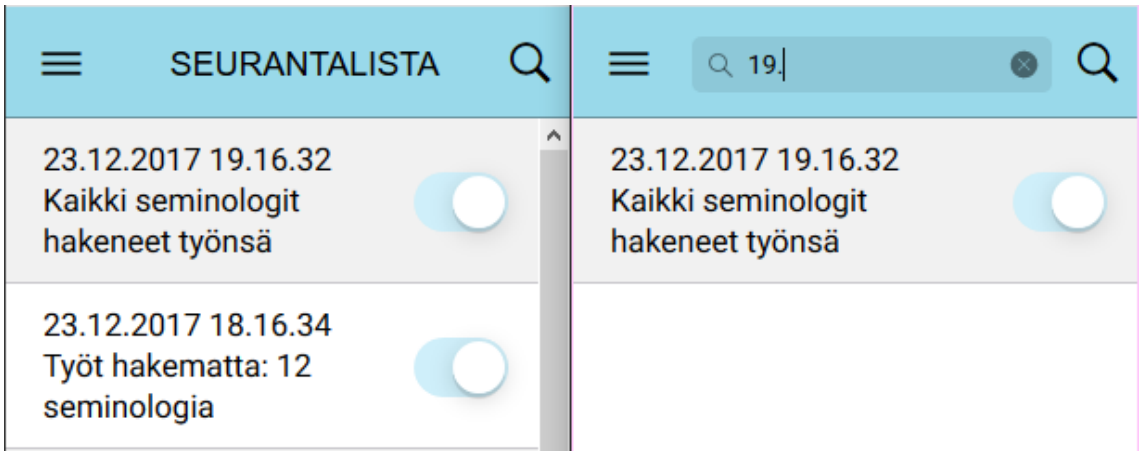
Toinen muutos oli suodatuksen lisääminen asetusvalikkoon. Näin käyttäjä pystyy muuttamaan suodatusasetuksia suoraan poistamalla toggle-valinnan asetusvalikosta. Alkuperäisessä suunnitelmassa suodatus oli samalla sivulla kuin päivystysviestit (Kuvio 3). Ongelmana pienissä napeissa oli nappien painaminen puhelimella. Tietokoneella pystyi nappeja painamaan tarkasti hiiren avulla, mutta puhelimella tämä oli haastavaa painikkeiden pienen koon vuoksi.

Yksi ratkaisu tähän olisi voinut olla painikkeiden koon suurentaminen, mutta ongelmaksi olisi tullut viestin luettavuus puhelimella, sillä puhelimen näyttö oli rajatun kokoinen. Suodatus lisättiin valikkoon ja sivulle lisättiin hakupainike, jolla voidaan etsiä haluttua termiä (Kuviot 4 ja 6).



Kuvio 5: Päivystyssovelluksen uudistettu valikko ja navigointi

Hakupainike on aktiivinen vain siinä tapauksessa, jos käyttäjä painoi sitä. Hakupainike näytti hakupalkin, joka meni yläpalkin tekstinpäälle, joten se ei vienyt tilaa päivystysviesteiltä, vaikka se olisikin ollut aktiivinen. Hakupalkin pystyy piilottamaan painamalla hakupainiketta uudestaan. Tämä toiminto voidaan havaita, kun katsomme ensin kuvion vasenta puolta ja vertaamme sitä oikeaan puoleen. Vasen on alkutilanne ennen kuin käyttäjä on painanut hakuikonia tai painiketta, jonka jälkeen hakupalkki ilmestyy näytölle. Jos käyttäjä painaa uudelleen hakupainiketta, se piilotetaan. (Kuvio 6.)

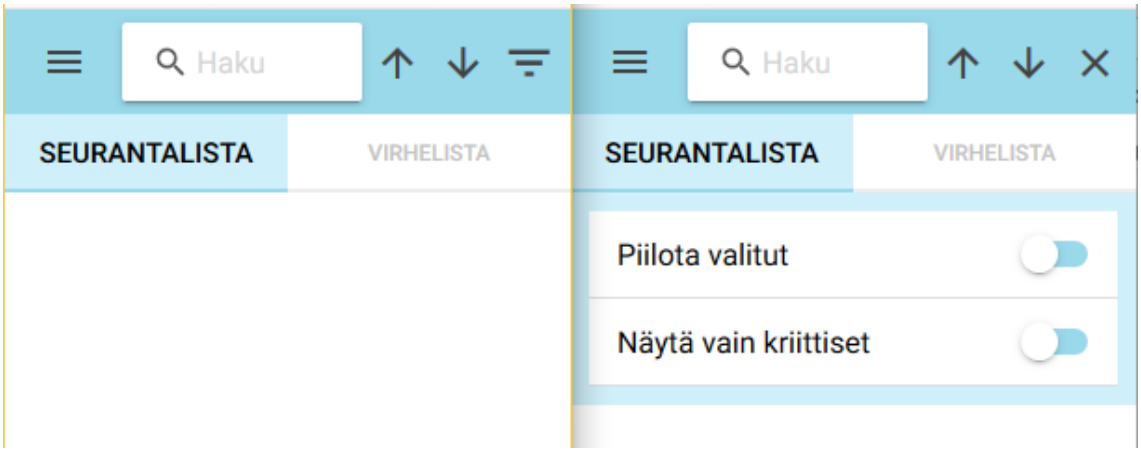


Kuvio 6: Listan suodatushaku

Jokaisella sivulla oli myös omat suodatusnapit, jotka vaikuttavat hakuominaisuuteen. Suodatusnapeilla voitiin näyttää tai piilottaa elementtejä valintojen mukaan. Suodatusnapit olivat jokaisella sivulla erilaiset, mutta ne voidaan kuitenkin koodata sivukohtaisiksi tai sivujen välisiksi napeiksi. (Kuvio 6.)

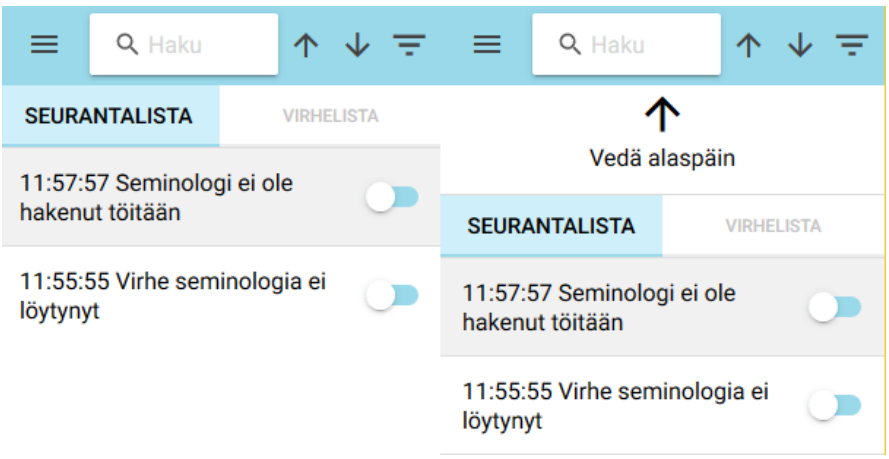
Käyttäjien kanssa käydyissä kokouksissa esille tulleiden toiveiden perusteilla sovelluksen suodatusta ja navigointia muutettiin sovelluksen valmistumisen jälkeen. Sovelluksen toiminnot toimivat oikein kuitenkin jo aiemmassa versiossa (kts. Kuviot 4 ja 7). Sovelluksessa on enää kaksi sivua, sovelluksen kirjautumissivu ja listasivu, joihin kaikki sivut tulevat välilehtinä. Suodatusnapit ovat jaettuja ja ne on mahdollista tehdä yksityisiksi per haluttu välilehti. Sovellus ei enää käytä pelkkää iOS-tyyliä, vaan se on käyttöjärjestelmä kohtainen. Jos vertaamme kuviota 7 kuvioon 6 huomaamme, että sovelluksen CSS-tyylit ovat muuttuneet enemmän Android-tyylisiksi (Kuviot 6 ja 7).

Kuvion 7 vasemmalla puolella näytetään suodatuspainikkeen oletustila. Oikea puoli kuviosta havainnollistaa mitä tapahtuu, kun käyttäjä painaa suodatusnappia. Suodatusnapin kohdalle ilmestyy ruksi, joka piilottaa suodatusnapit ruudulta. Hakupalkki ei ole enää piilotettavissa vaan se on aina näkyvissä. Sivun navigointi on suunniteltu helpottamaan ja nopeuttamaan sivujen vaihtoa. Hyötynä tässä muutoksessa on hakuominaisuuden toimivuus välilehtien välillä. Aktiivinen sivu on korostettu suuremmalla tekstillä kuin ei aktiivinen teksti. Sovellukseen lisättiin myös skrollaus-napit ylös ja alas. Tämän ominaisuuden tarkoitus on helpottaa tiedon selaamista varsinkin puhelimella. Käyttäjän painaessa ylöspäin nuolta käyttäjä ohjataan sovelluksen yläosaan, kun taas alaspäin nuolta painaessa käyttäjä ohjataan listanäkymän alaosaan. (Kuvio 7.)



Kuvio 7: Uudistettu navigointi ja suodatus

Tietojen hakuun suunniteltiin ominaisuus, joka mahdollistaa datajoukkojen haun milloin tahansa. Sovellus hakee ensimmäisen kerran käyttäjälle tuoreimman datan, mutta sen jälkeen datan päivitys on sovelluksen käyttäjän vastuulla. Kuviossa 8 näytetään miten käyttäjä voi hakea uutta dataa vetämällä ruutua alaspäin. Kuvion 8 vasemmalla puolella on sovelluksen oletusnäkymä, kun taas oikeassa puolella kuviota näytetään missä tietojen päivitystoiminto sijaitsee. Ominaisuuden tarkoituksena oli säästää tilaa käyttäjän näytöltä varsinkin puhelimella. Syynä ratkaisuun oli, ettei uuden data-painikkeen lisääminen veisi turhaa tilaa mobiilisovelluksessa, joten vetotoiminto oli tähän tarkoitukseen parempi, kuin esimerkiksi datajoukon haku-nappi. (Kuvio 8.)



Kuvio 8: Uuden datan haku

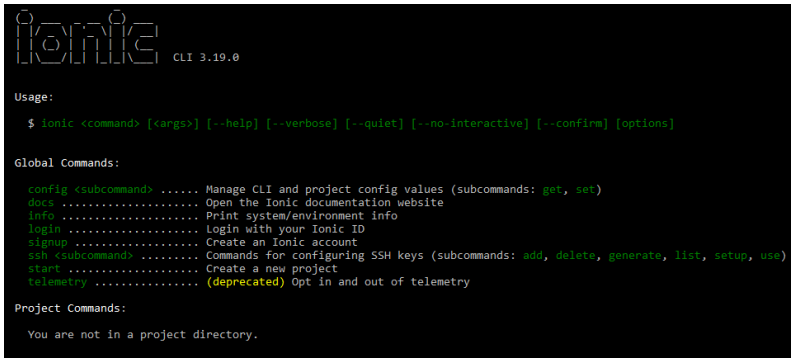
5.2 Sovelluskehitys

Kehitysluvussa kerrotaan, miten sovellus tehtiin, miten se aloitettiin, minkälaisia ratkaisuja tehtiin sovelluksen kehityksen aikana. Suurin osa kehityksestä keskittyi kuitenkin hybridisovelluksen toteutukseen.

5.2.1 Ionic CLI ja Ionic projektin päivitys

Ionic-projektin kehittämiseen voidaan käyttää muutamaa eri vaihtoehtoa. Yksi näistä vaihtoehdoista on Ionic CLI. Toinen vaihtoehto on tehdä Ionic projekti Visual Studiossa. Ionic CLI on riippuvainen Node.js -kirjastosta ja NPM-paketinhallinnasta.

Ionic CLI ei tarvitse oikeastaan muuta kuin Node.js:n ja NPM:n, joka tulee yleensä Node.js:n asennuksen mukana. Ionic CLI:n kautta voidaan myös luoda uusi projekti, sivuja tai komponentteja. Ionic CLI vaatii kuitenkin asentamisen NPM:n kautta ennen kuin sitä voidaan käyttää. Kuviosta 9 näemme miltä Ionic CLI näyttää komentorivillä (Kuvio 9).



Kuvio 9: Ionic CLI

Ionic CLI:ssä ei tosin voi luoda suoraan Ionic 2 -projektia vaan siihen on ladattava erikseen pohjat verkosta (Ionic 2 Templates for Visual Studio 2017). Vaihtoehtoina Ionic CLI tarjoaa Ionic 1- tai Ionic 3 -projektin. Ennen Ionic 3 -version julkaisua Ionic 2 -projekti voitiin luoda myös Ionic CLI:n kautta. Toisaalta ero näiden versioiden välillä on pieni, joka saattaa olla syy siihen miksi Ionic-tiimi oli päättänyt poistaa Ionic 2 -projektit Ionic CLI:stä.

Aluksi projektissa oli tarkoitus käyttää Ionic 1:stä, mutta se päivitettiin myöhemmin Ionic 2 -versioon, joka oli vanhaan versioon verrattuna nopeampi ja siinä oli uusia ominaisuuksia. Syynä tähän oli AngularJs:n päivitys uudempaan Angular versioon. Ionic 2 muutti myös tapaa, miten projekti rakennettiin. Koska kummastakaan sovelluskehittimen versiosta ei ollut juuri mitään tietoa, uudempaan sovelluskehittimeen siirtyminen ei ollut ongelma. Ionic 2 -projekti päätettiin päivittää Ionic 3 -projektiksi mahdollisten VirtualScrolliin kohdistuvien korjausten takia. Kyseinen vaihe todettiin hyödylliseksi, toteuttaa jo sovelluksen ensimmäisessä versiossa, mahdollisten lisäominaisuuksien takia. Ionic 2 -versiota ei enää nimittäin päivitetä Ionic-tiimin toimesta.

Sovelluksen Ionic-sovelluskehystä päätettiin päivittää vielä kerran, koska Ionic 3 ei eronnut paljon aiemmasta sovelluskehyksen versiosta. Ionic 3 mahdollistaa muun muassa lazy loading -ominaisuuden, joka nopeuttaa sovelluksen käynnistymistä ja lataa sivut ja komponentit vasta kun niitä tarvitaan ja sitä käytetään Ionic 3:ssa oletuksena. Tätä lähestymistapaa ei voi kuitenkaan käyttää kaikissa sovelluksissa. Ionic 3:ssa tämän ominaisuuden voi halutessaan ottaa pois.

Ionic 3:n toinen päivityssyy oli se, että VirtualScroll -ominaisuuden mahdolliset korjaukset tapahtuvat Ionic-tiimin toimesta. Päätimme sovelluksen käyttäjien kanssa poistaa VirtualScrollin käytön siihen liittyvien ongelmien takia. VirtualScroll oli yksi tapa näyttää suuria listoja puhelimilla optimaalisesti ilman, ettei sovellus tai sivu kuormittunut liikaa (VirtualScroll.)

Toinen tapa näyttää suuria listoja puhelimella oli InfiniteScroll. InfiniteScrollia voidaan käyttää myös VirtualScrollin kanssa. Toisaalta InfiniteScrollin heikkous oli se, ettei voinut koskaan tietää sivun kokonaisuuden kokoa vaan uutta dataa ladataan vasta, kun käyttäjä skrollaa aiemmin ladatun datan loppuun, jolloin navigointi hyppää ladatun datan verran. (InfiniteScroll.)

Toinen InfiniteScrollin heikkous oli se, ettei hakutilanteissa haeta tietoa kaikesta datasta vaan vain käyttäjän sillä hetkellä lataamasta datajoukosta. Tämä tarkoitti sitä, että jos kaikkea dataa ei oltu ladattu, ei käyttäjä voinut etsiä kaikkia hakutuloksia (Search lists of the whole data while using ionic Infinite scroll. 2015). Tämä saattaa vaikeuttaa sovelluksen käytettävyyttä. Sovellukseen oli mahdollista tehdä erillinen hakuratkaisu, joka etsii hakutulokset kaikesta datasta. Mikäli vaihtoehtoina oli VirtualScroll tai sivutus, joissa ei tarvitse tehdä erillistä hakuominaisuutta, ovat nämä teknologiat tai toteutustavat paljon houkuttelevimpia InfiniteScroll-tekniikan sijaan.

Kolmas tapa toteuttaa mobiilisovelluksen listan näyttäminen on sivutus. Sovelluksessa käytätettiin ngx-pagination komponenttia, joka mahdollistaa sivujen jaon haluttuun määrään. Esimerkiksi yhdelle sivulle halutaan 50 elementtiä, jaetaan sivu 50 elementin pätkiin. Tämä mahdollistaa optimaalisen ratkaisun puhelimella, koska tällöin sivu oli tarpeeksi nopea mobiiliselaimella. Testauksessa tosin ilmeni, että jopa 50 elementtiä oli liikaa ja tämän vuoksi määrä vähennettiin 25 elementtiin per sivu.

Suurin ongelma sivutuksessa oli puhelimen näytön koko. Esimerkiksi, jos painikkeita pitäisi saada mahtumaan 7 puhelimen näytölle ja niiden leveys olisi 25px ja niihin tulisi 5px välit oikealle ja vasemmalle, eivät painikkeet välttämättä mahtuisi puhelimen näytölle ja niitä olisi hankala painaa sormilla. (Kuvio 10.)

Haasteeksi tuli se, kuinka käyttäjälle voidaan näyttää x määrä sivuja niin, ettei sivujen vaihto ole liian vaikeaa tai turhauttavaa mobiililla ja verkkosivulla. Ratkaisuna ongelmaan lisäsin lisäominaisuuden ngx-pagination -ominaisuuteen. Lisäominaisuus mahdollistaa sivun vaihdon pyyhkäisy eleillä normaalin pagination sivunvaihdon lisäksi. Näin sovellus toimii myös pyyhkäisy liikkeistä ja linkkien painamisesta eli ominaisuutta voidaan käyttää verkkoselaimissa sekä puhelimilla. (Kuvio 10.)

Kuvio 10: Pagination/sivutus-ominaisuuden HTML-elementti

Tein HTML-elementtiin kuuntelijat, jotka lisäävät tai vähentävät sivunumeroa riippuen kumpaan suuntaan käyttäjä pyyhkäisee elementtiä sormella tai hiirellä. Vasemmalle pyyhkäisystä käyttäjä ohjataan seuraavalle sivulle, kun taas oikealle pyyhkäisystä käyttäjä ohjataan edelliselle sivulle, ellei sivu satu olemaan sivutuksen ensimmäinen sivu. Tässä tapauksessa pyyhkäisy ei tee mitään. (Kuvio 10.)

5.2.2 Visual Studio 2015 ja 2017 konvertointi

Ionic-projektit eivät välttämättä toimi suoraan Visual Studiossa vaan ne vaativat muutaman lisäosan lataamisen, jotta projekti toimii oikein. Visual Studio 2015 -versiossa projektin luominen on hieman monimutkaisempaa kuin Visual Studio 2017 -versiossa. Toimiakseen Visual Studio 2015 Ionic projekti vaatii:

- Visual Studio Tools for Apache Cordova -lisäosan Visual Studioon
- Visual Studio Ionic 2 Templates -projektipohjan (ei välttämätön, mutta suositellaan).
- NPM Task Runner -lisäosan
- Vähintään TypeScript 2.0.6 -version
- Microsoft ASP.NET and Web Tools Preview 2 -lisäosan Visual Studioon

(Get started with Ionic 2 apps in Visual Studio 2017.)

Projektissa Ionic 2 -pohja oli hyvä vaihtoehto, kun halusin luoda Ionic-projektin Visual Studio tai Microsoft ympäristöön. Toinen hyvä puoli Visual Studio 2015 -projektissa oli se, että sillä oli helppo tehdä Team Foundation Server eli lyhennettynä TFS-projekti.

TFS mahdollistaa Ionic-projektin lisäämisen muiden projektien joukkoon. Tämä oli kuitenkin pakollinen vaihe, koska yrityksessä käytetään pääosin TFS-versiohallintaa projektien testiin ja tuotantoon vietiin ja sillä säilötään koodia. Oletuksena Ionic-projekti käyttää Git-versiohallintaa, jota oli myös käytetty muutamassa yrityksen projektissa. Totesimme kuitenkin TFS-versionhallinnan turvallisemmaksi, jos se vain oli mahdollista tehdä Ionic-projektille.

Myöhemmin sovelluksen kehityksen aikana todettiin olevan järkevää konvertoida projekti Visual Studio 2017 -projektiksi. Hyötynä tässä on, että tulevaisuudessa kehittäminen on paljon helpompaa 2017- kuin 2015 -versiolla. Visual Studio 2017 ei myöskään vaadi niin paljon lisäosia tai säätöä kuin mitä 2015-versio vaati toimiakseen. Visual Studio 2017-ohjelma tarvitsee NPM Task Runner -lisäosan, TypeScriptin ja Node.js:n. Apache Cordovan asennus ei ole välttämätöntä, mutta jos sovelluksesta halutaan tehdä Android- tai iOS -versio, on Cordova asennettava.

5.2.3 Sovelluksen visuaalisen ja toiminnallisen osuuden toteutus

Sovelluksen visuaalinen toteutus koostui sivujen toteutuksesta ja Ionic 2-version dokumentaation opettelemisesta. Vaikeinta oli ymmärtää projektin alussa, miten komponentit toimivat ja miten sivun navigointi toimi. Onneksi Ionic-sovelluskehyksestä oli selkeä dokumentaatio ja sen pystyi opettelemaan suhteellisen nopeasti.

Toteutin sivut aluksi alkuperäisen suunnitelman mukaan (Kuvio 3). Visuaalisen toteutuksen jälkeen aloitin ohjelman toiminnallisuus-vaiheen. Sovelluksen toiminnallisuudesta kerrotaan tarkemmin kehityskappaleessa (kts. kappale 5.4.3). Jälkeenpäin ajatellen toiminnallisuuden toteuttaminen olisi ollut järkevämpää toteuttaa ennen visuaalista toteutusta, koska ne muuttuivat muutaman kerran projektin aikana.

Toistuvaa tai iteroitua kehitystapaa voitiin soveltaa myös sovelluksen kehityksen aikana. Oletetaan, että sovelluksen ensimmäinen versio oli prototyyppi. Kun prototyyppiä testattiin, se ei toiminutkaan niin hyvin tai siinä huomattiin käytettävyysongelmia, esimerkiksi turhia klikkauksia tai painalluksia, joita olisi voitu välttää. Suunnittelijan tekemä kuvio 3 oli siis tavallaan prototyyppi sovellukselle ja sitä parannettiin sovelluksen kehityksen aikana iteroimalla. (Nuutila ym. 2009, 204; Kuvio 3.)

Ajattelin itseni käyttäjän saappaisiin. Jos mielestäni käyttäjäkokemus oli huono, en toteuttanut ohjelmaa sen mukaisesti ja yritin keksiä ongelmaan ratkaisun. Ajattelin siis itseni päivystäjäksi. Varsinkin, jos joku ominaisuus olisi hidas tai se ei toimisi oikein, se antaisi huonon käyttökokemuksen ohjelmasta. Pahimmassa tapauksessa päivystäjät eivät käyttäisi sovellusta, jos sen käyttökokemus olisi liian huono. (Nuutila ym. 2009, 23-24.)

Sovelluksen kehityksessä käytettiin käyttäjäkeskeistä suunnittelua. Sovelluksen kehityksessä käyttäjäkeskeisyys näkyy siinä, miten usein palavereita järjestettiin käyttäjien kanssa. Palavereita oli melkein joka viikko ja niissä näytettiin tehtyjä muutoksia ja demottiin sovelluksen toimintaa. Tällä tavalla käytettävyyteen pystyttiin puuttumaan heti tuotteen kehityksen aikana. (Nuutila ym. 2009, 27-31.)

Ensimmäisenä sivuna toteutin kirjautumissivun. Kirjautumissivulla pystyin opettelemaan Ionic komponentteja ja Ionicin valmiita ominaisuuksia. HTML5-koodin yhteyteen sekoitettiin Ionicin luomia tageja. Ionic tagit nopeuttivat toiminnallisuuden toteuttamista. Esimerkiksi Ionicin SCSS muuttujilla pystyi määrittelemään ja muuttamaan Ionic sivun rakenteita. Myös jokaiselle sivulle oli erillinen tyylitiedosto. Tyylitiedostoon pystyi tekemään sivukohtaisia muutoksia.

Seuraavaksi toteutin navigointisivun. Oikeastaan sivun ainoa muutos oli luoda uudet CSS-tyylitykset suodatusnapeille. Eli periaatteessa sivun toteutus oli suurimmalta osin pelkkää tyylitysten ja luokkien lisäämistä nappeihin. Nappien tarkoitus oli toimia suodatuspainikkeina seuraaville sivuille.

Avaa Seurantalista -napin oli tarkoitus avata sivu uudelle sivulle, joka oli alkuperäisen suunnitelman mukaan pääsivu sovellukselle, johon pääsiassa sovellus keskittyisi. Optimointipalvelimen tilaukset -napin oli myös tarkoitus mennä omalle sivulle, joka näyttäisi samalta kuin seurantalistasivu, mutta sivulle tulisi eri dataa kuin Seurantalista -sivuille.

Seuraavaksi rupesin toteuttamaan seurantalistasivua suunnitelman mukaisesti. Tein kolmannen uuden CSS-tyylityksen pienemmille napeille. Nappien oli tarkoitus olla yhteydessä navigointi sivujen suodatusnappeihin ja niitä pystyi painamaan navigointi sivulta tai seurantalista sivulta. Ainoa uusi nappi oli näytä kuitatut-nappi, mikä olisi aktiivinen oletuksena ja sen voisi halutessaan kytkeä pois. Tällöin käyttäjä voisi piilottaa haluamansa viestit, joita hän ei halua enää nähdä.

Kun kaikki kolme sivua oli toteutettu suunnitelman mukaan visuaalisesti, siirryin tekemään sovelluksen toiminnallisuutta. Toiminnallisuuden toteutuksen aikana pohdin, mitä navigointinappi sisältää ja oliko takaisin -nappi oikeasti tarpeellinen. Lopulta päädyin yhdistämään navigointisivun ja seurantalistalla olevat suodatusnapit valikon taakse. Hyötyinä näin sen, ettei käyttäjän tarvitse mennä navigointisivulle joka kerta sovellukseen kirjautumisen jälkeen. Myös seurantalistalle jäi enemmän tilaa viestien tarkkailuun.

Koska käyttäjä joutuu selaamaan satunnaisen määrän rivejä riippuen tietokannasta haettujen rivien määrästä eli datasta, kannattaa suodatus ominaisuuden lisäksi sovelluksessa olla hakutoiminto. Aluksi suunnittelin lisääväni hakutoiminnon otsikon alle ja näin hakupalkki näkyisi sivulla aina. Kuitenkin mietin parempaa tapaa viedä mahdollisimman vähän tilaa puhelimen näytöltä, jotta dataa voidaan näyttää mahdollisimman paljon. Ongelman ratkaisemiseksi päätin tehdä togglenapin, joka näyttää tai piilottaa hakupalkin. Hakupalkki tulee sivun otsikon päälle, kun se on aktiivinen. Togglenappi on hakuikoni, jota painaessa voidaan aktivoida hakutoiminto ja piilottaa se painamalla hakuikonia uudestaan (kts. Kuvio 6).

Hakutoiminnon toteutuksessa törmäsin muutamaan optimointiongelmaan, mutta ne saatiin ratkaistua muuttamalla ohjelman rakennetta. Hakutoteutusten ero oli huomattava sovelluksen käytettävyyteen nähden. Ensimmäinen haku kesti paljon pidempään kuin optimoidussa hakuominaisuudessa, jossa dataa hakiessa ei näkynyt viivettä edes isolla määrällä datarivejä.

5.2.4 Socket.IO ja SignalR palvelimen toteutus

Socket.IO toimii Node.js-palvelimella ja se voidaan asentaa NPM-pakettienhallinnan kautta. Socket.IO-kirjasto mahdollistaa kaksisuuntaisen kommunikaation sovelluksen ja palvelimen välillä. Socket.IO pystyy kuuntelemaan tietokantaan tehtyjä muutoksia ja näyttämään ne reaaliaikaisesti verkkosivuilla. SignalR toimii samalla periaatteella kuin Socket.IO ja mahdollistaa sovelluksen ja palvelimen reaaliaikaisen tiedonhaun. Tosin SignalR on toteutettu paremmin Microsoft ympäristöön soveltuvaksi. (Introduction to SignalR 2014.)

Päädyin kokeilemaan SignalR-tekniikkaa Socket.IO:n sijaan IIS- ja Node.js -palvelinten yhteensopivuuden takia. Socket.IO olisi käytännössä pitänyt kääntää C#-koodiksi tai käyttää kolmannen osapuolen toteuttamaa IISNode-lisäosaa. SignalR kuitenkin toimii suoraan IIS-palvelimilla, joten sen käyttö oli tässä tapauksessa parempi vaihtoehto. SignalR:n kanssa oli tosin myös ongelmia, joten se päätettiin siirtää reaaliaikaisuuden kehityksen sovelluksen mahdolliseen jatkokehitykseen. Sovelluksen tietojen haku toteutettiin Ionicin kehittämällä veto-ominaisuudella. (kts. Kuvio 8; Introduction to SignalR 2014.)

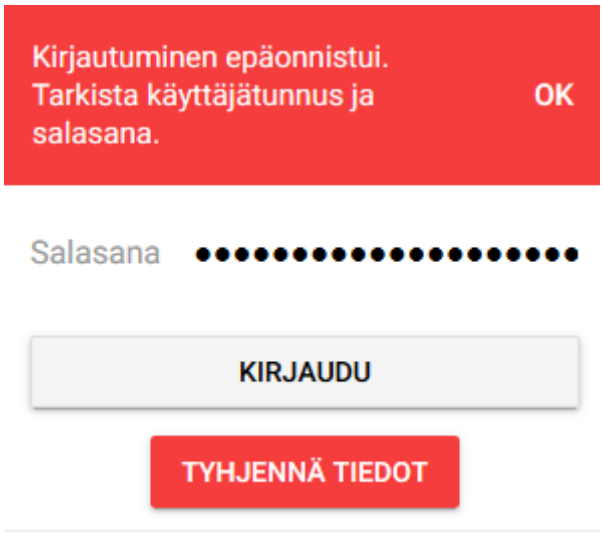
5.2.5 Sovelluksen autentikointi

Aluksi autentikointi oli kirjoitettu osaksi ohjelman koodia eli tietyllä salasanalla ja käyttäjätunnuksella käyttäjä pääsi sisään sovelluksen listausnäkymään. Tämä oli demo-ratkaisu, jolla testattiin miten se käytännössä toimisi sovelluksessa. Myöhemmin sovellukseen tehtiin oikea autentikointi käyttäjän tunnistamiseen.

Alun perin sovelluksen autentikointiin oli tarkoitus käyttää Oidc-client -kirjastoa, joka onnistumisen kirjautumisen jälkeen ohjaa käyttäjän takaisin ohjelmaan sisäänkirjautuneena. Selaimella tämä autentikointitapa olisi toiminut, mutta Oidc-client kirjaston heikkous oli se, että sovelluksella piti olla HTTP/HTTPS-osoite, johon käyttäjän pystyi ohjaamaan kirjautumisen jälkeen. Mobiilisovelluksella tämän kaltaista osoitetta ei ole. Sovelluksen käyttämä UIWebView ei käytä oikeaa verkko-osoitetta Android-käyttöjärjestelmässä. Sovelluksen navigointiosoite oli file:// alkuinen, eikä autentikointipalveluun voinut laittaa HTTP-osoitetta testausta varten. (VKWebView.)

Toisin kuin UIWebViewissä iOS-version VKWebViewissä oli oikeat verkko-osoitteet, joten periaatteessa käyttäjän uudelleen ohjaus onnistuisi suoraan sovelluksessa. VKWebView-tekniikka pitäisi testata, mutta testaukseen ei saatu Mac-tietokonetta, joten asian todentaminen oli mahdotonta. Ionicin mukaan VKWebView käyttää iOS:ssä localhost:8080 porttia, joka pitää autentikaatio palvelimen puolella myös sallia. Periaatteessa Oidc-clientin toteutus voisi toimia sovelluksen iOS-versiossa. (VKWebView.)

Päätimme kuitenkin parhaaksi toteuttaa sovellukseen kirjautumisen samalla sivulla ResourceOwner-tekniikalla, joka oli hieman erilainen autentikaatiotapa verrattuna Oidc-clientin tarjoamiin vaihtoehtoihin. Onnistuneen kirjautumisen jälkeen käyttäjältä tarkistetaan oikeudet palveluun. Jos käyttäjällä oli oikeudet palveluun, käyttäjä päästettiin sisään sovellukseen, muussa tapauksessa käyttäjälle näytettiin virheilmoitus Ionic toast-komponentin avulla, kuten voimme huomata alla olevasta kuvasta (kts. Kuvio 11).



Kuvio 11: Virheilmoitus kirjautuessa

Todennustapoja oli useita, mutta ainoa sovelluksen kanssa toimiva tapa oli ResourceOwner-tekniikka. ResourceOwner-tekniikka lähettää käyttäjätunnuksen ja salasanan identiteettipalvelimelle, joka taas palauttaa käyttäjälle tokenin, joka mahdollistaa sovelluksen käytön, jos käyttäjätunnus ja salasana ovat oikein.

5.2.6 Web-API

Sovelluksen web-rajapinta on toteutettu C#-koodilla. Rajapinnassa käytetään ASP.NET Core 2.0 -versiota. Sovelluksen tiedonhaku toteutettiin API-kutsuina. ASP.NET Core 2.0 -version hyötyinä verrattuna aiempaan versioon oli uusi raaka SQL-kysely, jota oli myös käytetty projektissa toteutetussa web-rajapinnassa (What’s new in ASP.NET Core 2.0. 2017).

Web-rajapinta haki kyselyillä uusimman datan, jota sovellus käytti. Tähän olisi mahdollisuus tehdä myös automaattinen datan päivitys sovelluksen ja rajapinnan välille, mutta päätimme jättää tämän toteuttamisen mahdollisesti sovelluksen versioon 2.0, jos se silloin on tarpeellinen ja ajankohtainen.

Sovellus kutsuu rajapintaa, joka hakee kyseisen päivän uusimmat datat kokonaisuudessaan. Sovellus parsii olemassa olevan datan perusteella mitä dataa pitää lisätä selaimen muistiin tai poistaa sieltä. Jos uutta tietoa löytyy, käyttäjän sovellukseen lisätään uusi rivi listalle. Jos käyttäjän hakiessa data ei vastannut API:sta palautettua dataa, päivitetään lista vastaamaan web-rajapinnan palauttamaa dataa. Yleensä näin käy, jos käyttäjä hakee uutta dataa seuraavana päivänä, jolloin selaimen välimuistiin oli jäänyt vanhaa dataa.

Aiemmin web-rajapinta oli toteutettu vanhemmalla ASP.NET Core-versiolla. Kuitenkin web-API päätettiin päivittää Core 2.0 -projektiksi, jotta SignalR Core-versiota voitaisiin testata. Tämä yritys kuitenkin kaatui SignalR Core alfaversion toimimattomuuteen sovelluksen kanssa. Alfaversio ei ollut tarpeeksi vakaa ja sen ominaisuudet eivät olleet vielä täysin valmiita sovelluksen vaatimaan käyttöön. SignalR ei pystynyt tekemään kahden sovelluksen yhdistämistä, kuten sen ASP.NET 4 -versio. SignalR:n ASP.NET Core versio on spekuloitu julkaistavan ASP.NET Core 2.1 version mukana vuonna 2018 (Daniel Roth. ASP.NET Core 2.1 roadmap. 2018).

5.3 Testaus

Sovelluksen testaus tapahtui käytettävyystestauksena, jonka sovelluksen kehittäjä, käyttäjät ja testaaja tekivät. Kehittäjä testasi sovelluksen toimivuuden muutosten jälkeen regressiotestauksella eli uudelleen testauksella, jossa pyritään tarkistamaan, onko aiempi virhe korjattu ja toimiiko se myös, jos ohjelman koodia muutetaan (Tuominen 2016, 23-24). Kehittäjän toimesta testaus tehtiin lasilaatikko -menetelmällä, koska kehittäjällä oli tieto ohjelman toiminnasta ja sen kehityksestä ja myös pääsy kyseisen ohjelman koodiin (Tuominen 2016, 22-23). Kun taas käyttäjät ja testaaja tekevät testauksen mustalaatikko -menetelmällä, jossa testaajalla ei ole pääsyä ohjelmistorakenteeseen tai tietoa, miten ohjelma oli toteutettu (Tuominen 2016, 21-22).

5.3.1 Testiversion luonti puhelimelle ja verkkosivuille

Testaukseen käytettiin Android-versiota Ionic sovelluksesta. Ionic Cordova loi APK-tiedoston, jonka pystyi asentamaan kännykälle USB-johdon kautta. APK-paketti pystyttiin luomaan Android Studiolla tai komentorivin avulla (Sign Your App). Sovelluksesta olisi voitu luoda myös iOS-versio, mutta siihen tarvittiin Mac-tietokone ja x-code kirjasto. Jos sovellus halutaan virallisesti Applen sovelluskauppaan, pitää ohjelma myös hyväksyttää Applella. Lisäksi käyttäjällä pitää olla Developer-tili, joka maksaa 99 dollaria vuodessa (Apple Developer Program 2017). Googlella on sama käytäntö, mutta maksu on 25 dollarin kertamaksu (Play Consolen käyttö).

Ionicissa on ominaisuus, jonka avulla Ionic luo www-kansion aina uudelleen, kun sovellus rakennetaan. Jos kansio on jo olemassa, sen sisältö päivitetään vastaamaan uusinta rakennettua sovellusta. Tuon saman www-kansion sisällön voi siirtää suoraan IIS-palvelimelle ja sovellus toimii ilman ylimääräisiä muutoksia. Projektista tehtiin TeamCity asennus, joka mahdollisti koodiin tehtyjen muutosten automaattisen viemisen testauspalvelimelle.

Sovellus toimi hyvin puhelimella, mutta ainoana negatiivisena asiana voisi mainita virheiden ilmoituksen. Tietokoneen selaimilla sovellus ilmoitti virheistä normaalisti. Tämä on luultavasti Ionicin lisäominaisuus, joka ilmoittaa ohjelman kaatumisen syyn. Eli puhelimelle pitää tehdä oma virheenhallintajärjestelmä, jotta virheet näkyisivät. Tämä ei sinänsä ole ongelma, mutta jatkossa olisi hyvä, jos tämä olisi otettu huomioon Ionicin ominaisuuksissa.

Toisena ongelmana huomasin, ettei Androidilla näkynyt ollenkaan vierityspalkkia. Tämän vuoksi käyttäjän oli vaikea hahmottaa, kuinka paljon rivejä yhdellä sivulla on. Selaimessa vierityspalkki näkyy normaalisti. Tämä oli Ionic-tiimin tekemä korjaus, koska Androidilla vierityspalkin näyttäminen aiheutti ongelmia, joten Ionic-tiimi päätti poistaa sen näyttämisen Androidilla. Android-selaimella vierityspalkki näkyi oikein, joten ongelma näyttäisi esiintyvän vain Android-sovelluksessa.

Huomasin myös vierityspalkin näkymisen vain kuin sovelluksen valikko oli aktiivinen, joka oli mielestäni erikoista. Mobiiliselaimissa tätä ongelmaa ei ilmennyt. Keskustelimme käyttäjien kanssa ongelmasta ja sen kriittisyydestä. Totesimme käyttäjien kanssa, ettei tämä ongelma vaikuta ohjelman toiminnallisuuteen niin paljoa, että sitä olisi järkevää korjata.

5.3.2 Testauksessa löydetyt ongelmat

Testauksessa ongelmia tuotti eniten Ionic-komponentti nimeltä VirtualScroll. VirtualScroll oli optimoinnin kannalta järkevin ratkaisu, koska komponentti lataa vain tarvittavan määrän elementtejä per sivu ja käyttäjän skrollatessa alaspäin vaihtaa elementtejä niin, että samat elementit päivittyvät vanhojen päälle. Sivujen lataamien elementtien määrä ei muutu vaan pysyy samana. (Taulukko 1.)

Valitettavasti VirtualScroll ei toiminut kaikkien komponenttien kanssa. Hyviä esimerkkejä tästä on tietojen filtteröiti ja hakuominaisuus. Jokaisessa Ionic versiossa oli joitakin bugeja, jotka vaikeuttavat sovelluksen käyttöä ja joissakin tapauksissa tekivät siitä vaikeampaa käyttää, kuin ilman VirtualScroll-ominaisuutta.

Ionic versio, jossa bugi ilmenee	**Löydetty bugi/ongelma**	**Korjattu?**
Ionic 2:sta alkaen	VirtualScroll kadottaa näkymän, jos näkymää selataan liian nopeasti.	Ei
Ionic 3:sta alkaen	Filtteröinti ei toimi oikein VirtualScrollin kanssa (visuaalisia ongelmia)	Ei
Ionic 3:sta alkaen	Filtteröinti ei hae kaikkea dataa, jos käytetään InfiniteScrollia	Ei
Ionic 2.3.0	VirtualScroll dataa ei ladata ensimmäisellä näkymällä.	Kyllä Ionic 3.1.0 lähtien
Ionic 2.2.0	Jos VirtualScroll näkymä ollaan skrollattu tietyn elementtien verran, se kadottaa näkymän, jos dataa piilotetaan filteröimällä tai painikkeiden avulla.	Kyllä Ionic 2.3.0 lähtien

Taulukko 1: Ionic-sovelluskehyksestä löydetyt ongelmat

Keskustelimme käyttäjien kanssa palavereissa, miten ongelmat kannattaa korjata. Päätimme kiertää ongelmat poistamalla VirtualScrollin käytön käyttämällä hieman muokattua sivutusta myös Android-käyttöjärjestelmässä. Suurin syy tähän oli se, että jokaisessa Ionic-versiossa Ionic 2-version jälkeen oli jonkinlaisia ongelmia, kuten taulukosta 1 voidaan havaita (Taulukko 1).

Sovelluksesta löytyi myös virheitä, jotka eivät olleet Ionicista riippuvaisia. Taulukossa 2 on kuvattu löydetyt bugit, jotka eivät liittyneet Ionic-sovelluskehykseen. Toinen korjaus on ehditty toteuttaa, kun taas toinen korjaus odottaa tarkempaa testausta ja vianselvittelyä. Sovimme yrityksen kanssa, että toisen bugin ehtii korjata myöhemmin, selainmuisti ongelma ei ollut niin kriittinen virhe kuin esimerkiksi VirtualScrollin aiheuttaman visuaaliset ongelmat. (Taulukko 2.)

Löydetty bugi/ongelma	**Korjattu?**
Jos seuraavana päivänä haetaan dataa ja selainmuistissa on vanhaa dataa voi datajärjestys olla väärinpäin uutta dataa hakiessa.	Kyllä
Uusien rivien palautus näytölle ei tapahdu heti, vaan käyttäjän on vaihdettava seurantalista- tai virhelistanäkymää nähdäkseen päivittyneet rivit sovelluksessa. Ongelma/bugi liittyy selainmuistiin.	Ei

Taulukko 2: Muut löydetyt ongelmat

5.4 Tuotanto

Sovellus tullaan viemään tuotantoon testauksen jälkeen. Koska sovelluksen julkaisu odottaa vielä tietokantaan tulevaa tietoa ja muutamaa sovelluskorjausta, ennen kuin se voidaan viedä tuotantoon, on tähän kappaleeseen kuvattu mahdolliset tuotantoon vientiin vaikuttavat asiat, jotka tulee ottaa tuotantoon viennissä huomioon. Kappaleessa kerrotaan myös sovelluksen toiminta, todistetaan responsiivisuus ja näytetään miltä lopullinen sovellus näyttää kokonaisuudessaan.

5.4.1 Sovelluksen vienti tuotantoon

Testauksen aikana tein projektille rakennusasetukset TeamCityyn. Tällä tavalla pystyttiin viemään palvelimelle tuotanto-versio testiin, joka kerta kun sovellukseen tuli muutoksia, se myös päivittyi TeamCityn kautta suoraan testipalvelimelle. Samalla periaatteella toimii myös tuotantopalvelin, mutta TeamCity asetukset ovat siinä hieman erilaiset.

Sovellus toimii kuitenkin samalla tavalla kuin testiversio. Oikeastaan ainoa muutos mitä pitää projektiin tehdä, on muuttaa Web-API osoitteet tuotantopalvelimeen viittaaviksi. Tietenkin myös oikeudet käyttäjille pitää lisätä tuotantoon. Projektin www-kansio viedään tuotantopalvelimelle, kun sovellus on valmis.

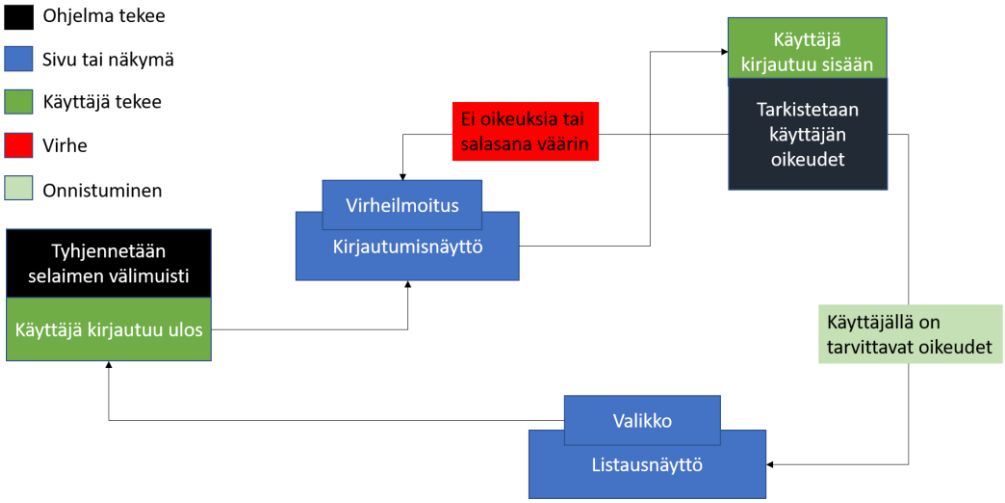
5.4.2 APK-paketin generointi ja allekirjoitus

APK-paketti generoidaan komentoriville kirjoitetulla tietyllä komennolla. Periaatteessa riittää, että komentorivi avataan projektikansiossa, jolloin APK-paketti rakennetaan sovelluksen www-kansion perusteella. Ennen projektin APK-paketin generointia sovellus yleensä rakennetaan uudestaan, APK-paketti oli oletuksena allekirjoittamaton ja se pitää allekirjoittaa ennen kuin sitä voidaan käyttää Androidissa.

APK-paketti voidaan myös luoda suoraan Android Studion kautta ja sen voi tehdä automaattisesti sitomalla APK-paketin kirjoitus omaan projektiin. On olemassa tapoja, jolla nämä vaiheet voidaan automatisoida ja paketti voidaan generoida, jokaisesta sovellukseen tehdystä muutoksesta luodaan siis aina uusi APK-paketti.

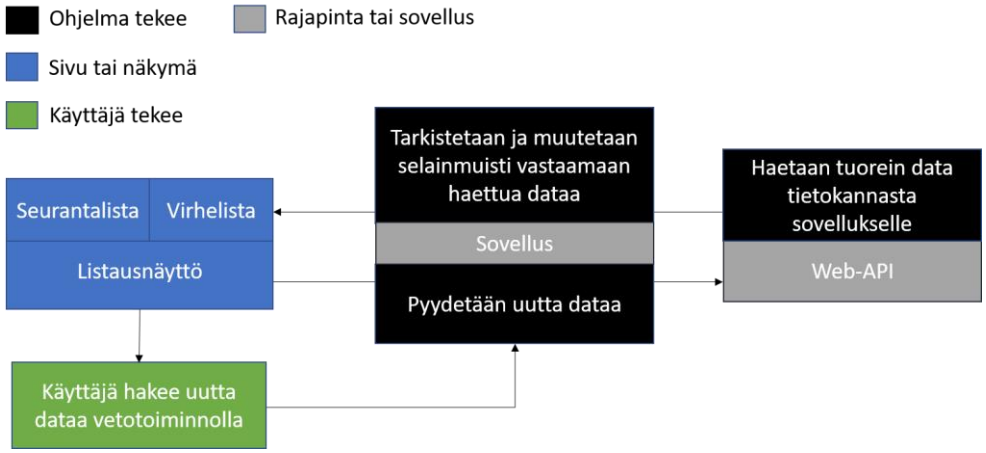
5.4.3 Sovelluksen toiminta

Käyttäjän on kirjauduttava ensimmäisellä kerralla sisään ennen kuin sovellusta voidaan käyttää. Jos käyttäjä ei syötä oikeita tietoja kirjautumiskenttiin, annetaan käyttäjälle virheilmoitus. Jos käyttäjällä on oikeudet palveluun ja kirjautumistiedot ovat oikeat, luodaan sovellukseen kirjautumissessio ja käyttäjä päästetään sisään. Käyttäjän on mahdollista kirjautua ulos sovelluksesta valikon kautta listausnäytöllä. Kuviossa 12 on kuvattu ohjelman toiminta, kun käyttäjä kirjautuu sisään tai ulos (Kuvio 12).



Kuvio 12: Sovelluksen sisään- ja uloskirjautuminen

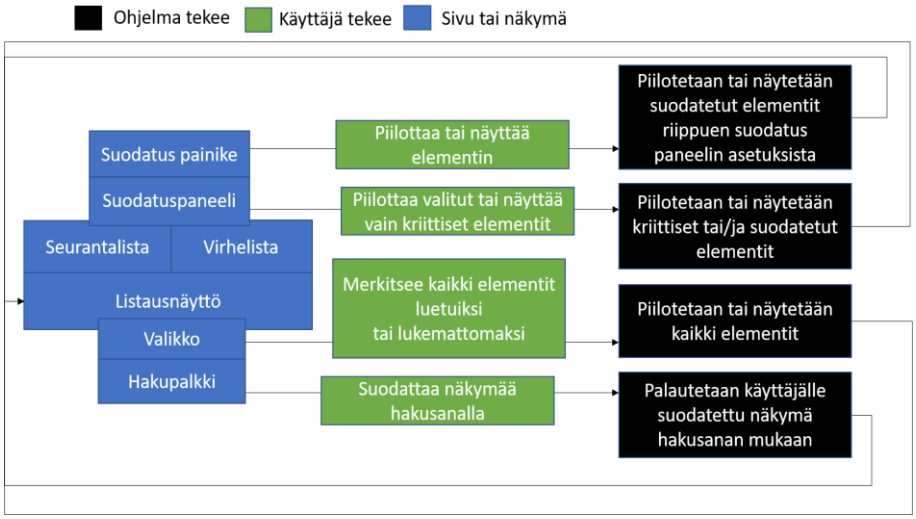
Onnistuneen kirjautumisen jälkeen käyttäjä viedään aloitussivulle eli listausnäytölle, jolta hän voi valita seurantalistan ja virhelistan. Sovellus hakee ensimmäisellä kirjautumisella uudet datat automaattisesti, jos selainmuistissa ei ole vielä ladattua dataa. Käyttäjällä on myös mahdollisuus hakea uutta dataa vetämällä seurantalistan ja virhelista tekstien kohdalta elementtiä alaspäin. Tätä kutsutaan kuviossa 13 vetotoiminnoksi. Kuviosta näkyy pelkistetysti datahaun periaate (Kuvio 13.)



Kuvio 13: Sovelluksen datan hakeminen

Listausnäytöllä on myös hakupalkki, jota voidaan käyttää hakutulosten suodatukseen. Listan elementeillä on myös painike, jonka avulla käyttäjä määrittää onko elementti valittu ja sitä kautta suodatettavissa erillisten valintojen avulla. Käyttäjän on mahdollista kirjautua ulos, tyhjentää sovellus, valita kaikki seurantalistan tai virhelistan valinnat aktiivisiksi tai poistaa kaikki valinnat. Listaus sivulla on myös mahdollista suodattaa listausnäkymää suodatusvalintojen avulla (Kuvio 14).

Tarkemman toiminnan sovelluksen ja käyttäjän välillä voi katsoa kuviosta 14. Kuviosta 14 näemme, kuinka käyttäjällä on mahdollisuus suodattaa seurantalista- tai virhelista -näkymää suodatus painikkeen ja suodatuspaneelin kautta. Listausnäytöltä käyttäjä voi suodattaa sovelluksen listausnäkymää valikon ja hakupalkin avulla (Kuvio 14).



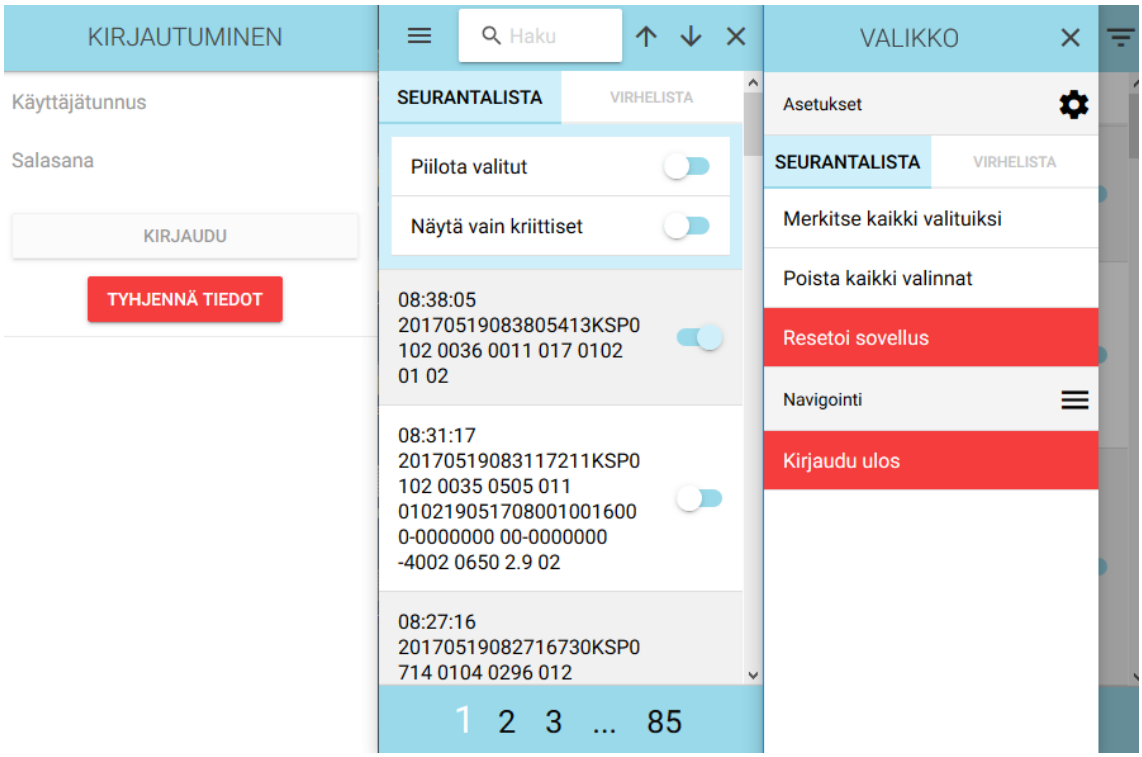
Kuvio 14: Sovelluksen suodatus

Seuraavasta kuviosta 15 saa paremman käsityksen sovelluksesta ja sen toiminnasta. Kuvion 15 vasen näyttö on kirjautumissivu. Kuvion 15 keskellä on listausnäkymä, jonne kirjautumisen jälkeen käyttäjä ohjataan ja kuvion oikealla näkyy sovelluksen valikko näkymä (Kuvio 15).

Kirjautumisnäytön päätarkoituksena on mahdollistaa käyttäjien kirjautuminen ja oikeuksien tarkastus ennen ohjelman käyttöä. Käyttäjän on mahdollista myös tyhjentää tietonsa painamalla tyhjennä tiedot -nappia.

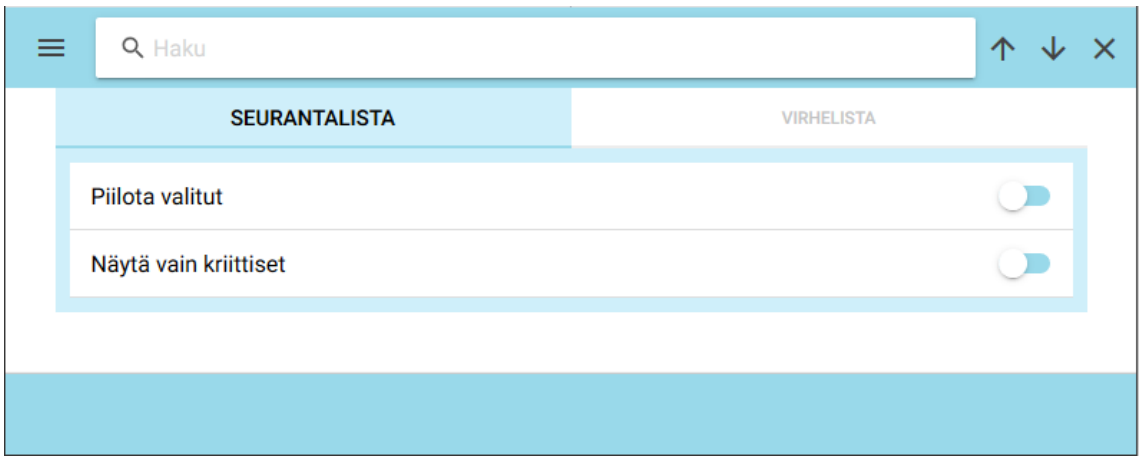
Listausnäyttö taas jakautuu kahteen eri näkymään seurantalistaan ja virhelistaan. Näkymien lisäksi se rakentuu valikkonapista, hakuominaisuudesta, skrollausnapeista, suodatuspaneelista, tietokannasta haetuista elementeistä ja sivutuspaneelista. Seurantalista- ja virhelistanäkymää on mahdollista vaihtaa halutessaan painamalla ei aktiivista painiketta, jolloin sivun tietojen näkymä vaihtuu (Kuvio 15).

Valikko taas mahdollistaa toimintoja, joita ei käytetä kovin usein, mutta ne helpottavat käyttäjän elämää huomattavasti. Tällä hetkellä sovelluksessa on mahdollista valita kaikki elementit luetuiksi tai poistaa valinta aiemmin valituista elementeistä. Jos kyseistä asetusta ei olisi, pitäisi käyttäjän valita tai poistaa valinnat kaikista suodatusnapeista yksitellen. Valikosta on myös mahdollista resetoida sovellus ja kirjautua ulos (Kuvio 15).



Kuvio 15: Sovelluksen sivut ja valikko Android-käyttöjärjestelmä

Kuviosta 16 voimme havaita, kuinka sovellus skaalautuu myös tietokonenäytölle. Sivun skaalautuvuuden mahdollistaa Ionic Grid HTML-tagi, joka toimii samalla idealla kuin Bootstrap Grid ominaisuus (Kuvio 16). Ionic Grid ja Bootstrap Grid -ominaisuus perustuu Flexbox-tekniikaan. Flexbox-tekniikka mahdollistaa elementtien järjestyksen sen mukaan paljon tilaa tietokoneen tai kännykän näytöllä on, joka helpottaa CSS-tyylitysten tekoa sovellukseen. Jos vertaamme kuvioita 15 ja 16 toisiinsa huomaamme, että Ionic Grid on muuttanut elementtien kokoa ruudulle sopiviksi (Ionic Grid 2017; Kuviot 15 ja 16).



Kuvio 16: Ionic Grid

6 Jatkokehitys

Jatkokehityksessä ja pohdinnassa käydään läpi mahdollisuuksia sovelluksen 2.0 version mahdolliseen toteutukseen. Erityisesti esille nousevat reaaliaikaisuus, käyttäjien ehdottamat uudistukset ja parannusehdotukset.

6.1 Reaaliaikainen tiedonhaku

Reaaliaikaisen tiedonhaun palvelimelta Ionic-sovelluskehykseen on mahdollista toteuttaa SignalR tai Socket.IO -kirjastoilla. Samoin kuin Socket.IO SignalR mahdollistaa reaaliaikaisen kommunikoinnin palvelimen ja käyttäjän välillä. Eroina kirjastoissa on se, että Socket.IO on JavaScript-kirjasto, kun taas SignalR on C#-kirjasto. Varmaan on olemassa muitakin kirjastoja, jotka mahdollistavat reaaliaikaisuuden, mutta nämä kirjastot tutkimisen jälkeen tulivat opinnäytetyön aikana yleisimmiksi tai järkevimmiksi toteuttaa.

6.2 Sovelluksen iOS versio

Sovelluksesta olisi vielä hyvä tehdä iOS-versio. Vaikka nykyinen sovellus toimii iOS-käyttöjärjestelmässä hyvin ilman natiivisovellusta web-selaimen kautta, olisi hyvä julkaista hybridisovellus myös iOS-käyttöjärjestelmälle. iOS-version tekoa saattaa rajoittaa Apple sovelluskaupan käyttö. Lisäksi apin julkaisuun tarvitaan ainakin 99 dollarin developer-lisenssi, jotta sovellus edes voitaisiin julkaista iOS-käyttöjärjestelmälle (Apple Developer Program). Ongelmana on myös se, ettei Applen sovelluskauppaan voi julkaista yksityisiä sovelluksia.

Applella on olemassa yksityisyysongelmaan ratkaisu, jolla yrityksen sisäiseen käyttöön voidaan julkaista sovelluksia. Ainoa ongelma tässä on se, ettei tällä lisenssillä voida julkaista versioita kaikille ja Enterprise Developer-lisenssi maksaa 299 dollaria vuodessa. Apple Developer Enterprise -ohjelmassa on myös muita ominaisuuksia, esimerkiksi testaus (Apple Developer Enterprise Program.)

Myös Ionic-tiimi tarjoaa sovelluksen yksityisyysongelmaan oman ratkaisun, jolla periaatteessa voitaisiin jakaa keskeneräisiä Android ja iOS -sovelluksia testaajille ja käyttäjille ilman, että niitä julkistetaan Android tai iOS -sovelluskauppaan. Ionic Pro sisältää myös palautetoiminnon suoraan sovelluksessa. Tähän kaikkeen tarvitsee ladata Ionic View -niminen sovellus, joka on tällä hetkellä saatavilla Android ja iOS -käyttöjärjestelmille.

Yksi vaihtoehto olisi myös käyttää kolmannen osapuolen sovelluksia ide-tiedoston generoitiin ja allekirjoitukseen ilman Mac-tietokonetta. Ainoa kyseenalainen asia näissä on se että, kyseiset sovellukset vaativat Apple ID:n sovelluksen allekirjoitukseen. Tämän jälkeen sovellusta voidaan käyttää iOS-laitteilla, kunhan Apple laitteen asetuksista kyseisen sovelluksen käyttö on sallittu.

6.3 Muut kehitysideat

Ionic on toteuttanut Stencil -työkalun, jolla voidaan luoda komponentteja, joita voidaan käyttää HTML-koodin kanssa. Sinänsä tekniikka on mielenkiintoinen, koska esimerkiksi projekti ei olisi enää riippuvainen yhdestä sovelluskehyksestä vaan periaatteessa komponentit toimisivat HTML-koodissa minkä tahansa sovelluskehyksen kanssa (Stencil: A Compiler for Web Components.)

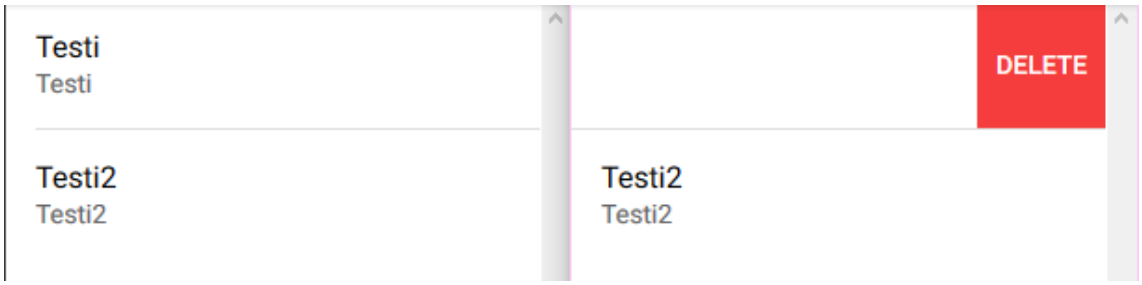
Toisena kehitysideana on sovellus, jossa hakuominaisuuksia voisi lisätä ja tallentaa selaimen muistiin. Tämä on mahdollista toteuttaa Ionic Storagen avulla, mutta vaatii suunnittelua siitä, mihin toiminto sijoitetaan sovelluksessa. Asetuksiin voisi laittaa erikseen esimerkiksi plus-napin, joka avaa modaalin suodattavia hakurajauksia varten. Nämä tiedot muistetaan ulos kirjautumiseen asti.

Toinen vaihtoehto on lisätä sama ominaisuus hakupalkkiin. Esimerkiksi käyttäjä voisi erotella hakutermit pilkulla ja oletuksena käyttäjän hakutermit tulevat mukaan hakuehtoihin, ellei niitä ole aikaisemmin poistettu. Tässä tapauksessa on kuitenkin luovuttava eri sivujen yhteisestä hausta, koska haettu tieto voi vaihdella välilehtien mukaisesti.

Navigointiin voisi lisätä extra ikkunan lisätiedoilla. Koska seurantasivuille ei mahdu kaikkea tietoa tietokannasta, voisi tämän ongelman ratkaista näyttämällä ylimääräisen sivun, joka näyttäisi valitun elementin tekstin uudessa ikkunassa tietokannan sisältämän datan. Tämä voisi sisältää reaaliaikaisen näkymän haetun datan kaikista riveistä. Silloin ei tarvitsisi mennä tarkistamaan tietoa tietokannasta, vaan tiedot löytyisivät suoraan käyttöliittymästä. Käyttöliittymässä voisi olla lisätiedot-nappi, joka vedetään esiin pyyhkäisyllä.

Käyttäjällä oli ehdotus, että sovelluksella voisi hakea myös yli päivän vanhoja rivejä tietokannasta. Tähän pitäisi tehdä erillinen haku, joka hakisi rivejä käyttäjän syöttämän päivämäärien väliltä. Käyttäjä voisi hakea sieltä haluttuja rivejä oikealle listalle ja ne tallennettaisiin selaimen muistiin. Myös listalla voisi olla ominaisuus, jossa lista-elementtiä vedettäessä vasemmalle tulee näkyviin tallenna-nappi, josta käyttäjä on mahdollista valita tallennustoiminto. Tallenna -napista tallennettaisiin tieto erilliselle listalle, esimerkiksi suosikit-välilehteen.

Esimerkkinä toimii kuvio 17. Kuviossa on elementtejä, joita voidaan vetää vasemmalle. Kun elementtiä vedetään vasemmalle, ruutuun tulee näkyviin poistonappi. Kuvan 17 tarkoitus on havainnollistaa uudistuksen mahdollisuudet. Vasen kuva on alkutilanne ja oikealla oleva kuva on tilanne elementin vasemmalle pyyhkäisyn tai vedon jälkeen. Myös nappeja voi olla useita tai oikealle vedettäessä tulisi esimerkiksi muokkaa-painike. Ideana on säästää tilaa puhelimen näytöltä, tällöin tilaa jää enemmän tekstin näyttämiseen (Kuvio 17.)



Kuvio 17: Kehitysidea veto/pyyhkäisy ominaisuus päivystyssovellukseen

Sain myös lisäehdotuksia käyttäjiltä, miten sovellusta voisi parantaa. Ehdotuksista kaikki on toteutettavissa. Yksi näistä ominaisuuksista on hakutulosten merkitseminen tai korostaminen. Myös eri tekstit voisivat olla sovelluksessa värikoodattuja, jotta niitä olisi helppo lukea mikä kuuluu mihinkin ja niiden erottaminen helpottuu.

Sovelluksen seuraavassa versiossa on mahdollista toteuttaa uudenlainen tapa näyttää dataa. Mutta nämä ominaisuudet voidaan lisätä vielä sovelluksen ensimmäiseen versioon, jos käyttäjät sitä haluavat. Korostamisen voisi toteuttaa hakemalla kaikki tekstit listalta, jotka vastaavat muuttuvaa hakutermiä ja tehdä niille oma CSS-tyyli, jota käytetään, kun hakupalkkiin on kirjoitettu tekstiä.

Sovelluksen olisi hyvä hakea tietokannan palauttama datajoukko reaaliaikaisesti sen muuttuessa ja palautettava se sovellukseen myös reaaliaikaisesti. Reaaliaikaisuus voidaan toteuttaa monella eri tavalla. Pitää ottaa myös huomioon kuinka paljon reaaliaikaista tiedonhakua voidaan käyttää yrityksen palvelin puolella.

Reaaliaikaisuus voisi toimia kuin Chat-sovellukset, poikkeuksena ettei käyttäjä itse voi lähettää viestejä sovelluksessa vaan viestin lähettämisen hoitaa palvelin. Samalla muut käyttäjät voivat nähdä palvelimen lähettämät viestit, kunnes ne tyhjennetään seuraavana päivänä. Datan saaminen mahdollisimman nopeasti on tärkeää ja koska datan hakeminen tietyn aikavälein on raskasta palvelimelle. Tässä tapauksessa on hyödyllisempää kutsua funktioita tai metodeja vasta sitten kun niitä oikeasti tarvitaan, kuin aina esimerkiksi 10 sekunnin välein.

Mahdollisesti myös reaaliaikainen rivinmuokkaus olisi mahdollista SignalR tai Socket.IO tekniikoilla. Muutos voitaisiin tallentaa reaaliaikaisesti tietokantaan. Tätä tapaa voisi mahdollisesti käyttää muissakin tulevaisuudessa toteuttavissa sovelluksissa.

7 Yhteenveto

Opinnäytetyössä opin kuinka kehittää web-rajapinnan, joka toteutettiin uudella ASP.Net Core 2.0 -tekniikalla, joka hakee tietokannasta tietoja. Opin myös, kuinka Ionic-sovellus kehitetään ja miten mobiilikehitys toimii verrattuna verkkosivukehitykseen. Projektissa saatiin uutta tietoa eri Ionic versioista ja niiden eroista. Samalla opin Angular-sovelluskehyksen perusteita, joita voin myös hyödyntää yrityksen uusissa sovelluksissa. Vaikka projektissa ei saatu toimivaa SignalR tai Socket.IO tiedonhakua, opin paljon niiden toiminnasta, mitä voidaan hyödyntää tulevaisuudessa erilaisissa asiakasprojekteissa.

Suurimmat ongelmat projektin toteutuksen aikana oli VirtualScroll-ominaisuuden käyttöönotto ja reaaliaikainen tiedonhaku. Datanhaku yritettiin toteuttaa reaaliaikaisesti, mutta se osoittautui liian työlääksi toteuttaa annetussa aikataulussa, joten reaaliaikaisesta tiedonhausta luovuttiin. VirtualScroll-ominaisuus ei myöskään toiminut odotetulla tavalla ja se korvattiin sivutus-lisäosalla ja muokattiin sekä selaimella että puhelimella toimivaksi vaihtoehdoksi.

Haasteellista oli sovelluksen autentikoinnin toteutus mobiililaitteille. Mobiililaitteet eivät tukeneet Oidc-client kirjastoa ja päädyimme käyttämään ResourceOwner-tekniikkaa sovelluksen autentikointiin.

Sovellus mahdollistaa helpomman virheiden seurannan yrityksessä toimiville päivystäjille. Projektin lopputuloksena saatiin toimiva sovellus yritykselle ja mahdollinen jatkokehityskohde. Samalla yritykseen saatiin lisää Ionic-kokemusta. Uskon, että projektista saatua kokonaiskokemusta voidaan hyödyntää tulevaisuudessa asiakkaiden kanssa ja yrityksessä päivittäisissä vuorovaikutustilanteissa työkavereiden kanssa.

8 Pohdinta

Projektin kehityksessä oli muutamia ongelmia, esimerkiksi miten Socket.IO saadaan toimimaan palvelin puolella ja miten tietokantapalvelimen kyselyt voidaan suorittaa tietyn väliajoin. Myönnän tehneeni virheen aloittamalla Socket.IO tekemisen varmistamatta soveltuuko se yrityksen palvelinympäristöön. En myöskään huomioinut autentikaation mahdollisia ongelmia, jotka aiheuttivat projektin viivästymistä aikataulusta. Aikataulumuutokset olisi voinut välttää tutkimalla tarkemmin teknologioita ja keskustelemalla jo projektin alkuvaiheessa työntekijöiden kanssa, jotka tietävät yrityksen tietokannoista ja palvelimista enemmän.

Lopulta tulin sokeaksi mihin Socket.IO oikeasti pystyy. Oikeastaan Node.js-sovelluskehystä pystyy pyörittämään IIS-ympäristössä esimerkiksi NodeIIS-projektin avulla, mutta on otettava huomioon, että se on kolmannen osapuolen ylläpitämä ja tämän takia riski. Jos NodeIIS tuki lopetetaan ei projekti välttämättä enää toimi. Myös tietoturva ongelmia saattaa tulla NodeIIS:n mukana.

Sovellus oli tarkoitus toteuttaa niin, ettei käyttäjän tarvitse tehdä mitään saadakseen uusi data, vaan data päivittyy automaattisesti. Olisi hienoa saada SignalR ja CLR Trigger- tai SQL Dependency -tekniikalla toteutettu reaaliaikainen tiedonhakudemo aikaiseksi, jotta voitaisiin kokeilla sen toimintaa käytännössä. On toisaalta otettava myös huomioon, kuinka usein päivityksiä tietokantaan tulee.

Tietyn väliajoin haettaessa oli ongelmia datan saamiseen liittyen. Välillä oma kone kuormittui täysin. Tämä johtui siitä, että sekä client että palvelin hakivat tietyn väliajoin uutta dataa samalla tietokoneella kaksi kertaa turhaan ja täten kuormittivat tietokoneen melkein kokonaan. Testissä tilanne oli toisin, koska palvelin hoiti serveripohjaisen tiedon haun, ei oma tietokone.

En ollut aiemmin toteuttanut Ionic projektia tai mobiilisovellusta. Suhtauduin tähän projektiin innolla ja mielestäni oli hauskaa opetella uutta. Kun aloitin projektin aluksi tutkin, Ionic 1 dokumentaatiota ja myöhemmin projektin aikana rupesin vertaamaan Ionic 1 versiota Ionic 2-versioon, jossa oli niin paljon parannuksia aiempaan versioon verrattuna, joten keskustelin työntekijöiden kanssa siitä, miksi projekti haluttiin toteuttaa Ionic 1:llä ja onko Ionic 2-version käytössä ongelmia. Syyksi ymmärsin, että yrityksellä oli enemmän tietoa vanhemmasta Ionic-versiosta ja Ionic 2-versiosta. Lopulta päädyin kuitenkin käyttämään projektissa Ionic 2-versiosta vanhemman version sijaan ja viimein konvertoimaan se Ionic 3-projektiksi.

Sovelluksen kehitys viivästyi kahdella kuukaudella suunnitellusta aikataulusta (Kuvio 2). Syynä viivästymiseen oli VirtualScroll ja reaaliaikaisen tietokantahaun toteutus. Kuitenkaan en pidä, muutaman kuukauden viivästymistä aikataulussa uhkana. Aikataulutuksen viivästyminen vain kertoo, ettei minulla ollut tarpeeksi kokemusta mobiilikehitys projekteista, että olisin osannut arvioida siihen kuluvan ajankäytön oikein. En ollut ottanut aikataulutuksessa huomioon ongelmia, joita voi tulla vastaan sovelluksen kehitysvaiheessa ja joiden ratkaisussa voi mennä paljon aikaa.

Sovelluksen kehityksessä olisi pitänyt aloittaa heti tietojen haulla tietokannasta ja vasta sitten tehdä front-end, johon keskityin aluksi ehkä vähän liikaakin. Samoin autentikointi olisi tullut tehdä aiemmin ja jättää sovelluksen front-end puoli viimeiseksi asiaksi.

Jos tekisin nopean demon Ionicista, tekisin ensin back-endin, joka hakisi datan sovellukseen. Tämän jälkeen tekisin sovelluksen autentikointiin tarvittavat muutokset. Muutosten jälkeen tekisin Ionic pohjan loppuun ja testaisin sovelluksen toimivaisuuden kokonaisuutena. Tämän jälkeen todentaisin toimivatko nämä ominaisuudet Ionicin kanssa ja miettisin kannattaako Ionic projektia käyttää. Samalla voitaisiin luoda testausprojekti puhelimelle ja verkkoselaimille nopeasti projektin käynnistyttyä.

Toisaalta kun on kerran tehnyt koodin toimivaksi seuraavalla kerralla tämän voi tehdä käyttämällä vanhaa koodipohjaa. Eli aika paljon vanhaa koodia voidaan hyödyntää uudelleen. Luultavasti ainakaan autentikaatio tai tietojen haku ei muutu hirveästi.

Projektin aikana opin reaaliaikaisen sovelluksen toteutuksesta, sen haasteista ja kehittymisestä. Opin myös, kuinka tehdä Ionic-projekti alusta loppuun. Sanoisin että Ionic toimii hyvin ainakin Firefox ja Chrome selaimilla iOS:n ja Androidin lisäksi.

Ongelmat joihin törmäsin projektin aikana, olivat minulle ennestään tuntemattomia. En ollut aikaisemmin miettinyt projektin toteutusta mobiilikehityksen näkökulmasta. Esimerkiksi kuinka näytän listan järkevästi mobiililla ja web-sivulla. Miten ratkaisen Ionic-sovelluskehyksen auttamat ongelmat, odotanko niiden korjausta Ionic-tiimiltä vai käytänkö toista toteutustapaa? Opinnäytetyön aikana opin paljon myös projektikokonaisuudesta. Esimerkiksi mitä sovelluskehityksessä pitää ottaa huomioon ja miten seuraavan sovelluksen kanssa kannattaa toimia asioiden helpottamiseksi.

Pakko myöntää olin hieman skeptinen uuden sovelluskehyksen opettelusta varsinkin, koska mobiilikehitys oli minulle kokonaan uusi asia. Kuitenkin Ionic-sovelluskehyksessä on hyödyllisiä ominaisuuksia, jotka toimivat mielestäni tarpeeksi hyvin. Tällä hetkellä ainoa poikkeus, on VirtualScroll.

Sovellus olisi voitu toteuttaa myös Xamarin ympäristössä. Xamarin on Microsoftin omistama sovelluskehys .NET ohjelmille. Huonona puolena Xamarin-sovelluskehyksessä on se, ettei sillä välttämättä pysty toteuttamaan nettisivuja suoraan kuten Ionicissa. Eli käytännössä pitäisi tehdä websivut ja mobiilisovellus projektit erikseen. Toisaalta on ehkä helpompi suunnitella ja erottaa ominaisuudet toisistaan, jos ne on tehty erikseen. Myös aikaa saattaa mennä enemmän kuin yhden sovelluksen toteutukseen kuluu aikaa.

SignalR opetteleminen vie aikaa varsinkin, jos halutaan sen palauttavan SQL-kyselyitä reaaliaikaisesti ja jos sen halutaan keskustelevan rajapinnan kanssa. Sama myös Socket.IO:n kohdalla. Molempien tekniikoiden täysi osaaminen vie aikaa, koska niiden ominaisuuksia pitää testata. Toisaalta ei ole varmaa onko reaaliaikaisuus perinteisen tavan sijaan vaivan arvoista, koska reaaliaikaisuus ei välttämättä toimi yrityksen tietokannan kanssa. Tietokantapalvelin ei välttämättä kestä reaaliaikaista tiedonhakua tai käyttäjien tarpeita ei voida täyttää yrityksen palvelimista johtuvien rajoitteiden takia. Tähän olisi hyvä tehdä demosovellus, jolla voitaisiin testata kuinka hyvin reaaliaikainen tiedonhaku toimisi yrityksen ympäristössä.

Socket.IO:n kehitys kannattaa varmaan tässä vaiheessa kuitenkin unohtaa ainakin, jos sovellus on tarkoitus ottaa käyttöön yrityksen sisällä IIS-palvelimen rajoitusten takia. Varsinkin, jos Node.js palvelin ei tue suoraan IIS-palvelinta tarkoittaisi tämä kolmannen osapuolen NodeIIS-palvelimen käyttöä.

Ennen projektin aloitusta rajapinnan tekeminen oli minulle uusi asia. Opin projektin aikana miten rajapinta toteutetaan, miten rajapinnasta haetaan dataa ja miten se soveltuu yrityksen palvelin ympäristöön. Sain sovelluksen hakemaan dataa rajapinnasta tietyn väliajoin. Olisin kuitenkin halunnut saada reaaliaikaisuuden toimimaan rajapinnan ja sovelluksen välillä. Toisaalta tämä toiminnallisuus olisi saattanut vaatia paljon enemmän aikaa.

Kun ASP.NET Core 2.0 -versio ilmestyi, olin hieman skeptinen sen käytöstä, koska siitä ei ollut hirveästi dokumentaatiota tai sitä ei juuri oltu ennen käytetty yrityksen sisällä. Opin kuitenkin miten uutta ASP.NET Core 2.0 voidaan hyödyntää toteutetussa rajapinnassa ja uskon että siitä on tulevaisuudessa hyötyä sekä minulle, että yritykselle.

VirtualScrollin käyttöönoton epäonnistuminen harmittaa, koska se olisi ollut optimoinnin kannalta todella hyvä ominaisuus. Opin kuitenkin projektin aikana kiertämään ongelmat, joita ilmeni VirtualScrollin yhteydessä. Uskon kuitenkin, että jossain vaiheessa Ionic-tiimi korjaa tai uudistaa VirtualScrollin niin että sitä on mahdollista käyttää isojen listojen näyttämiseen ilman visuaalisia bugeja ja muiden ominaisuuksien toimintaa.

Projektin lopuksi tein myös loppuhaastattelun sovelluksen käyttäjille, jonka tarkoitus oli selvittää, kuinka tyytyväisiä he olivat sovelluksen lopputulokseen ja projektin toteutukseen. Kyselyyn vastasi kaksi kuudesta käyttäjästä. Tarkkoja päätelmiä tästä ei voida tehdä, koska enemmistö käyttäjistä ei vastannut loppukyselyyn (Liite 2). Kuitenkin tulleista vastauksista voimme päätellä, ettei sovelluksen teko mennyt hukkaan ja sitä voidaan jatkossa parantaa tai kehittää.

Vastanneiden käyttäjien haastattelulla saatiin kokonaisuudessaan positiivinen kuva projektin etenemisestä ja käyttäjät kertoivat projektin lopputulokseksi toimivan ja suunnitellun mukaisen päivystyssovelluksen heidän tarpeisiinsa. Jatkokehitysideoita ei saatu paljon, mutta niitä oli jo aiemmin kysytty käyttäjiltä. Käyttäjien mielestä myös heidän mielipiteet ja muutostoiveet oli otettu huomioon sovelluksen kehitysvaiheessa, siltä osin mitä projektin laajuudessa oli mahdollista. (Sovelluksen käyttäjät 2017.)

Ohjelman jatkokehitys on käyttäjien mielestä myös mahdollista. Oikeastaan jatkokehitystarpeet selviävät vasta kun sovellus on otettu käyttöön. Yleensä projekteihin tulee vielä uusia ominaisuuksia sen valmistumisen jälkeen, sillä joskus käyttäjät huomaavat puuttuvia ominaisuuksia vasta kun sovelluksen suunniteltu osuus on toteutettu. Jatkokehitys ideana oli myös muiden tietojen hakujen lisäämistä sovellukseen, jotka helpottaisivat päivystäjän työtä. Laajempia hakuominaisuuksia olisi yksi vastanneista halunnut mukaan jo sovelluksen kehityksessä. Kuitenkaan projektiin käytettävä aika ei välttämättä olisi riittänyt hakuominaisuuden laajentamista reittioptimointiin, yksittäisen asiakkaan tietojen hakuun tai muuhun vastaavaan toimintaan. Toisaalta sovelluksessa on jo hakutoiminta, jolla pystyy etsimään tietoa, joten halutun ominaisuuden teko ei olisi välttämättä mahdotonta jatkokehityksessä. (Sovelluksen käyttäjät 2017.)

Omasta näkemyksestäni sovellus saatiin toteutettua, vaikka kaikkia toimintoja ei ehditty toteuttamaan suunnitellussa aikataulussa tai kaikkia ominaisuuksia ei saatu toimimaan sillä tavalla kuin olisin halunnut. Tämä on kuitenkin osa sovelluksien kehittymisprosessia. Sovelluksen jatkokehitys on mielestäni myös mahdollista tulevaisuudessa varsinkin, koska teknologiat kehittyvät nopeasti. Teknologian kehittyminen mahdollistaa uusien ominaisuuksien tekemisen sovellukseen.

Itse sovelluksen kehittäjänä näen käyttäjien kanssa käytyjen palavereiden auttaneen sovelluksen lopputulokseen pääsemisessä. Toivoin myös mahdollisessa jatkokehityksessä tiivistä yhteystyötä käyttäjien kanssa. Tällöin on mahdollista saada paras hyöty sovelluksen jatkokehityksestä, eikä turhia ominaisuuksia toteuteta sovellukseen, joka vie resursseja muualta.

Oli vaikeaa arvioida käyttäjien lopullista mielipidettä sovelluksesta, koska sovellus ei ole täysin vielä tuotannossa. Saamme kuitenkin, jonkinlaista käsitystä siitä oliko projekti tuottanut hyötyä yritykselle ja käyttäjille. Ainakin itse opin nopealla tahdilla toteuttamaan mobiilisovelluksia ja opin miten suunnittelua voidaan tehdä mobiiliympäristössä, jota voin myöhemmin hyödyntää työelämässä. Oppimia taitoja voi myös hyödyntää Ionic-sovelluskehyksen ulkopuolella esimerkiksi front-end puolella. Uskon myös rajanpinnan kehittämisen oppimisen auttavan myös back-end toteutuksissa tulevaisuudessa.

Lähteet

Painetut

Airaksinen, T. & Vilkka, H. 2003. Toiminnallinen opinnäytetyö. Helsinki: Tammi.

Hyvärinen, M., Nikander, P. & Ruusuvuori, J. (toim.) 2017. Tutkimushaastattelun käsikirja. Tampere: Vastapaino.

Nuutila, E., Sinkkonen, I. & Törmä S. 2009. Helppokäyttöisen verkkopalvelun suunnittelu. Helsinki: Tietosanoma.

Sähköiset

About Node.js. Viitattu 25.11.2017. https://nodejs.org/en/about/

Apple Developer Enterprise Program. Viitattu 11.11.2017. https://developer.apple.com/programs/enterprise/how-it-works/

Apple Developer Program. Viitattu 11.11.2017. https://developer.apple.com/programs/how-it-works/

ASP.NET SignalR. Viitattu 5.8.2017. http://signalr.net/

CLR Triggers. 2017. Viitattu 3.12.2017. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/clr-triggers

CLR Triggers. Viitattu 13.8.2017. https://msdn.microsoft.com/en-us/library/ms131093(SQL.100).aspx

Continuous Integration with TeamCity. 2016. Viitattu 31.12.2017. https://confluence.jetbrains.com/display/TCD9/Continuous+Integration+with+TeamCity#ContinuousIntegrationwithTeamCity-WhatisContinuousIntegration

Creative Bloq Staff. When to paginate and when to infinite scroll. 2015. Viitattu 10.3.2018. https://www.creativebloq.com/ux/paginate-or-infinite-scroll-71515816

Daniel Roth. ASP.NET Core 2.1 roadmap. 2018. Viitattu 10.3.2018. https://blogs.msdn.microsoft.com/webdev/2018/02/02/asp-net-core-2-1-roadmap/

Get started with Ionic 2 apps in Visual Studio. 2017. Viitattu 13.8.2017. http://taco.visualstudio.com/en-us/docs/tutorial-ionic2/

InfiniteScroll. Viitattu 10.3.2018. https://ionicframework.com/docs/api/components/infinite-scroll/InfiniteScroll/

Intro to SCSS for Sass Users. 2017. Viitattu 26.11.2017. http://sass-lang.com/documentation/file.SCSS_FOR_SASS_USERS.html

Introduction to ASP.NET Core. 2017. Viitattu 3.12.2017. https://docs.microsoft.com/en-us/aspnet/core/

Introduction to SignalR. 2014. Viitattu 5.8.2017. https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Introduction to the C# Language and the .NET Framework. 2015. Viitattu 2.12.2017. https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Ionic 2 Templates for Visual Studio 2017. Viitattu 24.2.2018. https://marketplace.visualstudio.com/items?itemName=VisualStudioPlatformTeam.Ionic2TemplatesforVisualStudio2017

Justyna Rachiwicz. When, How And Why Use Node.js Your Backend. 2017. Viitattu 18.3.2018. https://www.netguru.co/blog/use-node-js-backend

Play Consolen käyttö. Viitattu 17.12.2017. https://support.google.com/googleplay/android-developer/answer/6112435?hl=fi

Sass (Syntactically Awesome StyleSheets). 2017. Viitattu 26.11.2017. http://sass-lang.com/documentation/file.SASS_REFERENCE.html

Search lists of the whole data while using ionic Infinite scroll. 2015. Viitattu 18.3.2018. https://forum.ionicframework.com/t/search-lists-of-the-whole-data-while-using-ionic-infinite-scroll/26393

Sign Your App. Viitattu 7.1.2018. https://developer.android.com/studio/publish/app-signing.html

Stencil: A Compiler for Web Components. Viitattu 7.1.2018. https://stenciljs.com/docs/intro/

Team Foundation Server. Viitattu 30.12.2017. https://www.visualstudio.com/tfs/

TypeScript. Viitattu 7.1.2018. https://www.typescriptlang.org/

What is npm? 2017. Viitattu 25.11.2017. https://docs.npmjs.com/getting-started/what-is-npm

What's new in ASP.NET Core 2.0. 2017. Viitattu 19.11.2017. https://docs.microsoft.com/en-us/aspnet/core/aspnetcore-2.0

VirtualScroll. Viitattu 10.3.2018. https://ionicframework.com/docs/api/components/virtual-scroll/VirtualScroll/

VKWebView. Viitattu 10.12.2017. https://ionicframework.com/docs/wkwebview/

Julkaisemattomat

Sovelluksen käyttäjät. 2017. Palautekysely sovelluksen käyttäjien kanssa 5.12.2017. IT-alan yritys.

Suunnittelija. 2017. Suunnittelijan haastattelu 28.11.2017. IT-alan yritys.

Tuominen, A. 2016. Ajanhallinta- ja raporttityökalun testaus. Viitattu 19.11. https://www.theseus.fi/bitstream/handle/10024/120263/Tuominen_Anna.pdf?sequence=1

Kuviot

Taulukot

Liitteet

Liite 1: Suunnittelijan haastattelu

Minkälainen oli projektin määrittelyvaihe ja miten se tehtiin?

Mikä oli mielestäsi vaikeinta sovelluksen ulkonäön suunnittelussa?

Miten mobiiliohjelman suunnittelu eroaa verkkosivun suunnittelusta?

Milloin visuaalinen suunnitelma sovelluksesta tehtiin, arviolta?

Jos oli yli vuoden vanha suunnitelma, vastaako se nykyajan mobiilisovellusten standardeja?

Oliko sovelluksen värimaailmaan tietty syy, jos oli mikä?

Tein muutaman muutoksen alkuperäiseen suunnitelmaan, olivatko ne tarpeellisia, olisiko niistä tullut kommunikoida paremmin?

Voiko mielestäsi liian pitkä aikaväli suunnittelun ja toteutuksen välillä aiheuttaa ongelmia projektin aikataulussa tai lopputuloksessa?

Kuinka usein projektin alkuperäinen suunnitelma muuttuu projektin aikana tai uusia ominaisuuksia yleensä pyydetään suunnittelemaan, yleensä?

Onko suunnitelman muuttaminen suunnittelun jälkeen hyvä vai huono asia? Miksi?

Liite 2: Käyttäjien loppuhaastattelu

Saavutettiinko projektin toteutuksen aikana haluttu lopputulos?

Onko mielestäsi sovelluksen jatkokehitys tarpeellista?

Mitä jäi mielestäsi puuttumaan sovelluksen toteutuksesta?

Otettiinko käyttäjien toiveet/tarpeet mielestäsi huomioon projektin kehityksen aikana?